MASTER EN SCIENCES INFORMATIQUES

Logique du raisonnement analyse-synthèse-développement d'outils informatiques

Lorge, Freddy

Award date:
1985

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX

(NAMUR)

INSTITUT D'INFORMATIQUE

LOGIQUE DU RAISONNEMENT

ANALYSE - SYNTHESE - DEVELOPPEMENT

D'OUTILS INFORMATIQUES

Mémoire présenté par
Freddy LORGE
en vue de l'obtention du titre de
LICENCIE ET MAITRE EN INFORMATIQUE
Promoteur : J. BRUNIN
Année académique : 1984/1985

Remerciements

Je désire avant tout remercier le directeur de ce mémoire, Monsieur J. BRUNIN, pour sa documentation personnelle et ses idées nombreuses.

Je tiens à remercier Madame CEREGHETI et Madame NOEL pour la dactylographie : leur intervention fut rapide et efficace.

Enfin, je n'oublie pas tous ceux dont la collaboration anonyme m'a aidé à surmonter les difficultés rencontrées.

I N T R O D U C T I O N

Cet ouvrage retrace le cheminement de notre recherche sur la logique des raisonnements. Il s'agit d'une réflexion motivée par deux questions fondamentales : "Est-il possible d'estimer la vraisemblance d'un discours ?"
"Etant donné un énoncé, quelles vérités peut-on en déduire ?".

Plutôt qu'à la signification des propositions, nous nous sommes intéressés aux liens logiques qui les enchaînent. Car ce sont eux qui font l'unité d'un texte, qui le rendent cohérent ou inconsistant . Mal utilisés, ils peuvent introduire des ambiguités dans un exposé (contradictions, redondances, omissions). C'est ainsi que des propositions sont parfois présentées comme conséquences directes d'un énoncé avec lequel elles sont incompatibles. Aussi, le souci constant qui anime ce travail est-il le développement de méthodes pour tester la validité des conclusions.
Dans le même état d'esprit, nous avons cherché des procédés permettant de construire des conclusions de manière précise. Nous avons donc accordé une attention particulière à l'implication.
Le rôle privilégié qu'elle joue dans la logique nous autorise à l'utiliser comme moteur de la déduction : sans elle, nul raisonnement n'est possible.

Si la traduction d'un texte en formules mathématiques offre l'avantage d'une étude rigoureuse, elle comporte en contrepartie un inconvénient majeur : les expressions sont lourdes et compliquent singulièrement l'analyse.
C'est pourquoi nous avons voulu développer des algorithmes suffisamment généraux que pour examiner un grand nombre de propositions.
Ces algorithmes travaillent sur des informations directement interprétables par l'ordinateur, c'est-à-dire codées dans un langage binaire.

Le premier chapitre concerne l'algèbre booléenne : il donne les rudiments de la logique algébrique et constitue une introduction indispensable pour tout lecteur qui désire s'initier à la logique moderne.

Prolongement naturel à la logique algébrique, la logique des propositions détaillée au second chapitre est l'occasion de définir clairement ce qu'est une implication et un énoncé valide. Une procédure de test y est présentée, qui permet de résoudre les nombreux problèmes proposés.

Pourtant, l'application de cette procédure aux syllogismes classiques engendre des erreurs de jugement. Elles sont fort heureusement éliminées par la théorie de la quantification. Celle-ci ouvre la porte à des raisonnements plus complexes exposés au troisième chapitre.

Le quatrième chapitre est consacré à la théorie des relations.
Nous y traitons plus spécialement les relations univoques et décrivons la démarche qui nous a amené à créer un algorithme propre à découvrir dans un ensemble de relations, tous les sous-ensembles de relations compatibles entre elles.

En annexe enfin, nous avons conçu un programme sur base des trois premiers chapitres. Il est capable de résoudre de nombreux exercices et notamment de programmer les tables de décisions.

CHAPITRE I.

ALGEBRE BOOLEENNE

Nous donnons dans ce chapitre quelques éléments d'algèbre booléenne, nécessaires à la compréhension des chapitres suivants. Les bases indispensables sont concentrées dans le premier paragraphe. Le second traite, de façon très formelle, la simplification des fonctions booléennes; les notions importantes sont celles de consensus et d'implicants primitifs; elles seront redéfinies de manière moins académique au moment voulu.

1. NOTIONS FONDAMENTALES.

1.1. Variables booléennes.

Nous appellerons variable booléenne toute variable pouvant prendre les seules valeurs 0 ou 1.

a.- Postulats de base.

$0 \cdot 0 = 0$	$0 + 0 = 0$
$1 + 1 = 1$	$1 \cdot 1 = 1$
$0 \cdot 1 = 1 \cdot 0 = 0$	$0' = 1$
$1 + 0 = 0 + 1 = 1$	$1' = 0$

Ces postulats utilisent les opérateurs logiques + (disjonction),
. (conjonction) et
' (négation)

On peut les récapituler dans les tableaux suivants :

valeurs		+	.
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

valeurs	'
0	1
1	0

Nous noterons indifféremment x' ou $\bar{x}$ la négation de la variable x.

b.- Relations générales à 1 variable.

Soit x, 1 variable booléenne; on peut facilement montrer à l'aide des postulats de base que :

$$0 + x = x \qquad x + x' = 1$$
$$1 \cdot x = x \qquad x \cdot x' = 0$$
$$1 + x = 1 \qquad x + x = x$$
$$0 \cdot x = 0 \qquad x \cdot x = x$$

c.- Relations générales à 2 variables.

On peut démontrer les égalités suivants :

$$x + xy = x(1 + y) = x \cdot 1 = x$$
$$x(x + y) = xx + xy = x + xy = x$$
$$x(x' + y) = xx' + xy = 0 + xy = xy$$
$$x + x'y = x + xy + x'y = x + y$$

L'opérateur de négation obéira aux règles :

$$(xy)' = x' + y'$$
$$(x + y)' = x'y'$$

1.2. Fonctions booléennes.

Nous appellerons fonction booléenne toute fonction de n variables booléennes ($n > 0$) dont la valeur (0 ou 1) est déterminée par la valeur de ses composants.
La disjonction, conjonction et négation de une ou plusieurs variables booléennes sont des fonctions booléennes.
Une question se pose alors : les 3 opérations +, ., ' suffisent-elles à décrire toute fonction booléenne concevable avec n variables ? ($n > 0$)
La réponse est que la négation et la conjonction suffisent, sans même la disjonction.

En effet, soit 2 variables booléennes x et y :

le seul cas où $x + y$ doit valoir 0 est celui où x et y valent simultanément 0, c'est-à-dire où $(x'y')$ vaut 1 ;

au lieu d'écrire $(x + y)$, nous pouvons donc nier $x'y'$ et écrire $(x'y')'$.

Etant donné la description d'une fonction booléenne, c'est-à-dire étant donné une énumération montrant quelle valeur prendra la fonction pour chaque choix de valeur de ses composants, nous pouvons construire à l'aide de la conjonction et de la négation, une fonction répondant à la description.

Par exemple : soit F, 1 fonction booléenne de x, y, z décrite ainsi : F vaut 1 dans les cas :

x = 0	y = 1	z = 1
1	0	1
1	1	0
0	1	0
0	0	0

F vaut 0 dans les 3 cas restants :

x = 1	y = 1	z = 1
0	0	1
1	0	0

Les 3 derniers cas sont respectivement les cas où xyz vaut 1, x'y'z vaut 1 et xy'z' vaut 1.
La fonction que nous cherchons peut donc s'obtenir en niant simultanément les 3 cas non souhaités :

F = (xyz)'(x'y'z)'(xy'z')',
soit, après développement :
F = x'y + xy'z + x'z' + yz'

Cette méthode de construction ne s'applique pas si la fonction vaut 1 dans tous les cas; il faut un autre traitement.

Par exemple : si on veut exprimer F, 1 fonction de x, y, z,w qui vaut 1 dans tous les cas, on écrira :

$$F = (x\ x'\ y\ z\ w)'$$

En effet, x'x vaut 0 dans tous les cas et par conséquent (x x' y z w) vaut 1 dans tous les cas.

La conjonction n'est pas moins superflue que la disjonction; toute fonction booléenne peut également s'écrire en terme de disjonction et de négation:
en effet : la conjonction (x y) est transposable en (x' + y')'.
Il n'en reste pas moins que par souci de clarté dans l'écriture, il est utile d'introduire d'autres opérateurs.

1.3. Quelques autres opérateurs logiques.

Soient x, y et z, 3 variables booléennes.

a.- La disjonction exclusive.

x ou (exclusif) y, noté $x \oplus y$, prend la valeur 0 si, et seulement si, x et y ont les mêmes valeurs.
On a donc :

$$\begin{aligned} x \oplus y &= (xy)'(x'y')' \\ &= (x' + y')(x + y) \\ &= x'y + xy' \end{aligned}$$

b.- Le conditionnel.

Le conditionnel ' si x alors y', noté $x \supset y$, prend la valeur 0 si x a la valeur 1 et y la valeur 0.
On a donc : $x \supset y = (xy')' = x' + y$

c. - L'homologie.

L'homologie ' x si, et seulement si, y', notée $x \equiv y$, équivaut au double conditionnel $(x \supset y)(y \supset x)$ et vaut 0 ssi

x a la valeur 1 et y la valeur 0
ou y a la valeur 1 et x la valeur 0

On a donc : $(x \equiv y) = (xy')'(x'y)' = (x' + y)(x + y') = x'y' + xy$

1.4. Forme normale disjonctive et conjonctive.

On appellera forme normale disjonctive, toute fonction booléenne F qui possède les 3 propriétés suivantes :

(1) il n'y intervient que les 3 opérateurs +, ., '
(2) la négation est limitée à des variables simples
(3) la conjonction ne réunit que des lettres et des négations de lettres.

Exemple : F = x' y z + x y' z + x' y z'

Une forme normale conjonctive possédera les propriétés suivantes :

(1) il n'y intervient que les 3 opérateurs +, ·, '
(2) la négation est limitée à des variables simples
(3) la disjonction ne réunit que des lettres et des négations de lettres.

Exemple : $F = (x + y)\ (y' + z)$

Partant d'une forme normale disjonctive, nous pouvons la transformer en forme normale conjonctive en utilisant l'égalité :

$$a + bc = (a + b)(a + c)$$

La transformation inverse peut se faire par simple distribution du produit sur la disjonction.

1.5. L'implication.

Nous dirons qu'une fonction booléenne F implique une autre fonction G (et nous noterons $F \rightarrow G$) si, et seulement si, le conditionnel $F \supset G$ a la valeur 1 quelles que soient les valeurs des variables booléennes.

Exemple : $x\ y \rightarrow x$

car $\overline{xy} + x = x' + y' + x = 1$

2. LA SIMPLIFICATION DES FONCTIONS BOOLEENNES.

Il s'agit de trouver des formes $F_1(x_1, x_2 \ldots x_n)$ égales à une forme proposée $F(x_1, x_2, \ldots x_n)$ et ne contenant aucun terme superflu, c'est-à-dire dont la suppression laisserait F_1 inchangé.

Etant donné une forme normale disjonctive F, on peut :

I) Vérifier si un constituant quelconque de F est redondant.

En effet, sachant que :

X implique X' ssi $X + X' = X'$ (1)

($X \rightarrow X'$ ssi $\bar{X} + X' = 1$, ssi

ssi $\bar{X}\,\bar{X}' + X' = 1$

ssi $\bar{X}\,\bar{X}' = \bar{X}'$

ssi $X + X' = X'$)

Nous déduisons :

Si un constituant de F implique le reste de la fonction F, ce constituant est redondant.

Par exemple, dans la fonction

$\bar{p}\,\bar{s} + \bar{p}\,\bar{q} + \bar{s}\,q + \bar{r}\,p\,s$,

$\bar{p}\,\bar{s}$ est redondant car il implique $\bar{p}\,\bar{q} + \bar{s}\,q + \bar{r}\,p\,s$

II) Vérifier si un élément littéral est redondant dans un constituant X de F.

Cet élément sera redondant si le reste de X implique F.

En effet : Soit X_1, le reste de X (= X amputé de l'élement littéral

dont on peut vérifier la nature redondante)

par (1), X_1 implique F ssi $X_1 + F = F$

comme $X_1 + F$ contient $X_1 + X$

et $X_1 + X = X_1$

on a X_1 implique F ssi X_1 peut remplacer X dans F

Par exemple :

$pqr + p\bar{r} + \bar{q}\bar{r}$ est égale à la fonction plus simple

$pq + p\bar{r} + \bar{q}\bar{r}$ car $pq \rightarrow pqr + p\bar{r} + \bar{q}\bar{r}$

Ces deux méthodes de simplification ne suffisent pas à simplifier toute fonction.

Par exemple, aucune d'elles ne permet de simplifier

$$p\bar{q} + \bar{p}q + q\bar{r} + \bar{q}r$$

en la fonction équivalente

$$p\bar{q} + \bar{p}r + q\bar{r}$$

QUINE a proposé une méthode de simplification plus générale, que nous allons développer dans les pages qui suivent.

2.1. Produit fondamental.

Soit un ensemble de variables booléennes distinctes $(x_1, x_2, \ldots x_n)$ intervenant dans une fonction booléenne F.

Nous appellerons monôme ou produit fondamental, toute forme non nulle du type $X = \prod_i x_i \prod_y x'_j$ ($\prod$ = produit booléen)

c'est-à-dire tout produit non nul de variables distinctes, affirmées (x_i) ou niées (x'_j).

Nous pouvons écrire F sous forme normale disjonctive :

$$F = \sum_k X_k$$

où X_k sont des produits fondamentaux.

Nous noterons d'une lettre majuscule les produits fondamentaux et d'une lettre minuscule les variables booléennes.

Théorème 1.

Dans l'ensemble des produits fondamentaux différents formés à partir de n variables $(x_1, x_2, \ldots, x_n)$, on a :

$X \longrightarrow X'$ ssi $X = X'X_1$

démonstration

Sachant que :

$$X \longrightarrow X' \text{ ssi } X + X' = X' \qquad (1)$$

démontrons que la condition est suffisante :

si $X = X' X_1$

alors $X + X' = X' X_1 + X' = X'$

par (1), on a : $X \longrightarrow X'$

démontrons que la condition est nécessaire :

Si $X \longrightarrow X'$

par (1), on a : $X + X' = X'$

X' ne peut contenir de lettres n'appartenant pas à X car si x appartient à X' et non à X, la valeur de X + X', pour x = 0, dépendrait de X et non de X' identiquement nul.

Donc, toutes les lettres de X' appartiennent à X.
De plus, une lettre ne peut être affirmée dans X et niée dans X' car si x = 1 et si toutes les lettres de X sont égales à 1, on a X = 1 et X' = 0, incompatibles avec X + X' = X'.

Donc,toutes les lettres de X' appartiennent à X et sont affirmées ou niées de la même façon dans X et X'.

Donc, X est de la forme X' X_1.

On dira que X est plus long que X', ou que X est un multiple de X'.

Par exemple : X = abc'd
X'= ac'
on a : X $\longrightarrow$ X' et X = X_1 X', où X_1 = bd

remarque : Soient un produit fondamental et une forme normale disjonctive, X et F ne contenant pas la lettre x (ni x, ni x');

si x X $\longrightarrow$ F, alors X $\longrightarrow$ F.

En effet, x X $\longrightarrow$ F
avec x n'appartient pas à X
x n'appartient pas à F
en prenant x = 1, on a : X $\longrightarrow$ F

2.2. Implicant primitif.

On appelle implicant d'une forme normale disjonctive F, tout produit fondamental X qui implique F.

Par exemple : F = a + b + c
chacun des termes de la somme est un implicant de F car a + F = F.

On appelle implicant primitif d'une forme normale disjonctive F, tout produit fondamental X_1 qui implique F et n'implique pas d'autre produit fondamental impliquant F.

si pour tout X, $X_1 \longrightarrow X \longrightarrow F$

alors $X = X_1$

par exemple : Soit $X_2 = ab$ et $X_2 \longrightarrow F$

$X_1 = abc$ et $X_1 \longrightarrow F$

comme X_1 est multiple de X_2 ($X_1 \longrightarrow X_2$), X_1 ne saurait être un implicant primitif de F.

Théorème 2.

Si X est un implicant primitif de F, alors X a toutes ses lettres dans F

démonstration

Supposons que X contienne une lettre x n'appartenant pas à F.

On aurait : $X \longrightarrow X_1$ avec $X = x\, X_1$

comme on a $X \longrightarrow F$

$x\, X_1 \longrightarrow F$

on a d'après la remarque du paragraphe 1

$X_1 \longrightarrow F$

On aurait donc :

$X \longrightarrow X_1 \longrightarrow F$

Et X ne pourrait être un implicant primitif.

Il résulte du théorème précédent que le nombre d'implicants primitifs d'une forme normale disjonctive est fini.

Théorème 3.

X est un implicant primitif de F ssi X est un implicant de F et X ne contient aucune lettre redondante.

démontrons que la condition est suffisante :

$X \longrightarrow F$

il n'existe pas x tel que $X = X'x$ et $X' \longrightarrow F$

par le théorème 1 :

il n'existe pas X' tel que $X \longrightarrow X'$ et $X' \longrightarrow F$

donc X est un implicant primitif

démontrons que la condition est nécessaire :

Si X contenait une lettre redondante, on aurait :

$X = x\,X' \longrightarrow X' \longrightarrow F$

et X ne serait pas un implicant primitif.

Théorème 4.

$F = \sum_k X_k$ peut s'écrire $F = \sum_p X_p$ où on somme tous les implicants primitifs de F

démonstration

$F = \sum_k X_k$ chaque X_k est un implicant de F car $X_k + F = F$

après suppression dans $\sum_k X_k$ de toutes les lettres redondantes, on a :

$F = \sum_{k'} X_k'$ où chaque X_k' est un implicant primitif de F

(en vertu du théorème 3)

Tout implicant primitif X de F étant tel que $X + F = F$,

on peut étendre la somme sur X_k' à une somme sur tous les implicants primitifs de F

$$F = \sum_p X_p$$

2.3. Consensus.

- Deux produits fondamentaux X_1 et X_2 admettent un consensus ssi il existe une et une seule lettre qui soit affirmée dans un de ces produits fondamentaux et niée dans l'autre.

  par exemple :

  $x_1 x_2 x'_3$ et $x_1 x'_2 x_3 x_4$ n'admettent pas de consensus

  $x_1 x_2 x'_3$ et $x_1 x_2 x_3 x_4$ admettent un consensus.

  Le consensus (X_1, X_2) de X_1 et X_2, s'il existe, est le produit des formes déduites de X_1 et X_2 par suppression de la lettre opposée

  par exemple :

  $$(x_1 x_2 x'_3, x_1 x_2 x_3 x_4) = x_1 x_2 x_4$$

- Etant donné une suite S de produits fondamentaux; $S = (X_1, X_2, \ldots X_n)$, on peut lui associer la suite S' dérivée de S; S' = suite des consensus (X_i, X_j).

  L'ensemble des suites S et S' constitue une nouvelle suite S_1 de produits fondamentaux :

  $$S_1 = (S, S')$$

On peut former la suite S'_1, dérivée de S_1 et former

$$S_2 = (S_1, S'_1)$$

Le nombre des suites S_i est limité puisque le consensus de deux produits fondamentaux contient une lettre de moins que le nombre de lettres différentes contenues dans les deux produits fondamentaux.

- On appelle suite complète, notée S_c, la dernière suite ainsi formée.

L'ensemble des consensus associés à S est la réunion de suites dérivées

$$U\ S' = C \quad \text{et on a} \quad S_c = S\ U\ C$$

Théorème 5.

$$(X_1, X_2) \longrightarrow X_1 + X_2$$

démonstration

$$X_1 \longrightarrow X_2 \text{ ssi } X_1 \bar{X}_2 = 0$$

il suffit donc de démontrer que :

$$(X_1, X_2)\ \overline{(X_1 + X_2)} = 0$$

en désignant par x la lettre opposée dans X_1 et X_2, on a :

$$X_1 = x\ X'_1 \ ; \ X_2 = x'\ X'_2$$

$$(X_1, X_2) = X'_1\ X'_2$$

$$\overline{(X_1 + X_2)} = \bar{X}_1\ \bar{X}_2 = (x' + \bar{X}'_1)\ (x + \bar{X}'_2)$$

$$(X_1, X_2)\ \overline{(X_1 + X_2)} = X'_1\ X'_2\ (x' + \bar{X}'_1)(x + \bar{X}'_2) = 0$$

(tous les termes du développement sont nuls).

Généralisation.

$$((X_1, X_2), X_3) \longrightarrow X_1 + X_2 + X_3$$

En effet : $((X_1, X_2), X_3) \longrightarrow (X_1, X_2) + X_3$

avec $(X_1, X_2) \longrightarrow X_1 + X_2$

Par conséquent, tout produit fondamental de C implique la somme des produits fondamentaux de S dont il est issu et, a fortiori, il implique la somme totale $\sum_k X_k$

Théorème 6.

Tout implicant primitif X_p de $F = \sum_k X_k$ $(F \neq 1)$ appartient à la suite complète S_c associée à la suite S des X_k.

Supposons que X_p n'appartienne pas à S et montrons que X_p appartient alors à la suite C des consensus associés à S

démonstration

Considérons les produits fondamentaux X ayant les 3 propriétés suivantes :

(1) $X \longrightarrow X_p$

(2) $X \not\longrightarrow X_k$ pour tout k (X n'implique aucun X_k de F)

(3) X a toutes ses lettres dans F

X_p est un produit fondamental ayant ces trois propriétés :

le point (1) est évident

le point (3) est évident grâce au théorème 2

le point (2) : soit X_p, différent de X_k pour tout k, un implicant primitif

on a $X_p \not\longrightarrow X_k$ pour tout k

en effet, si $X_p \longrightarrow X_k$ avec X_p différent de X_k

X_p ne serait pas un implicant primitif de F,

puisqu'alors $X_p \longrightarrow X_k \longrightarrow F$.

- Soit X_o, un élément de l'ensemble X tel que :

si $X \longrightarrow X_o$, alors $X = X_o$

Pour tout X, il existe au moins une lettre x de S, n'appartenant pas à X.

En effet :

Si une forme X contenait toutes les lettres de S, X contiendrait toutes les lettres de X_k pour tout k; or, on ne peut avoir $X \longrightarrow X_k$ (propriété (2)). Donc X et X_k contiendraient au moins une lettre opposée et l'on aurait :

$$X \cdot X_k = 0 \quad \text{pour tout k}$$

d'où :

$$X \cdot \sum_k X_k = 0 \quad \text{ou} \quad X \cdot F = 0$$

Mais, d'après la proriété (1), on a : $X \longrightarrow X_p \longrightarrow F$

donc $X \overline{F} = 0$ ce qui n'est compatible avec $X F = 0$ que pour X identiquement nul (0 n'appartient pas à l'ensemble des X par la propriété (2)).

- En particulier, il existe donc au moins une lettre x_o de S n'appartenant pas à X_o. Mais alors, $x_o X_o$ et $x'_o X_o$ sont des produits fondamentaux plus longs que X_o (impliquant X_o) et satisfaisant aux propriétés (1) et (3); ils ne peuvent donc satisfaire à la propriété (2).

Autrement dit, il existe une forme X_1 et une forme X_2 parmi les X_k de S telles que :

(4) $\quad x_o\, X_o \longrightarrow X_1 \quad$ et $\quad x'_o\, X_o \longrightarrow X_2$

(avec $X_o \not\longrightarrow X_1$; $X_o \not\longrightarrow X_2$; x_o n'appartient pas à X_o).

Etudions les formes possibles de X_1 et X_2 :

- X_1 et X_2 contiennent respectivement x_o et x'_o d'après la remarque du paragraphe 1 : $x_o\, X_o \longrightarrow X_1$; donc si X_1 ne contient pas x_o, on aurait $X_o \longrightarrow X_1$, ce qui est faux.

- On ne peut avoir

$$X_1 = x_o \quad \text{avec} \quad X_2 = x'_o$$

car alors on aurait F = 1, ce qu'on suppose exclus.

- Donc 3 cas restent possibles :

(a) $X_1 = x_o\, X'_1$ et $X_2 = x'_o$

dans ce cas, (4) s'écrit $x_o\, X_o \longrightarrow x_o\, X'_1$

donc : $X_o \longrightarrow X'_1$ où l'on a $X'_1 = (X_1, X_2)$

(b) $X_1 = x_o$ et $X_2 = x'_o\, X'_2$

dans ce cas, $X_o \longrightarrow X'_2$ où l'on a $X'_2 = (X_1, X_2)$

(c) $X_1 = x_o\, X'_1$ et $X_2 = x'_o\, X'_2$

dans ce cas, (4) s'écrit :

$$x_o\, X_o \longrightarrow x_o\, X'_1 \quad \text{et} \quad x'_o\, X_o \longrightarrow x'_o\, X'_2$$

d'où $X_o \longrightarrow X'_1$ et $X_o \longrightarrow X'_2$

donc $X_o \longrightarrow X'_1 \cdot X'_2$

où l'on a $X'_1 X'_2 = (X_1, X_2)$

(il ne peut, en effet, y avoir de lettres opposées dans X'_1 et X'_2 car alors $X'_1 X'_2 = 0$, d'où $X_o \longrightarrow 0$ et $X_o = 0$)

° dans les 3 cas, on a (d'après le théorème 5)

$$X_1 + X_2 = X_1 + X_2 + (X_1, X_2) \text{ et } X_o \longrightarrow (X_1, X_2)$$

- Ainsi, si X_p n'appartient pas à S, on peut ajouter à F un consensus de la suite C, sans changer F; F se présente sous la forme F_1 :

$$F_1 = F + (X_1, X_2) = F$$

X_p est encore un implicant primitif de F_1 et s'il n'appartient pas à la suite S_1 des produits fondamentaux composants F_1, on pourra comme précédemment ajouter à F_1 un consensus de la suite C, sans changer F_1.

- L'ensemble des X soumis aux conditions (1), (2), (3) ne sera plus le même que précédemment car la condition (2), $X \not\longrightarrow X_k$ pour tout k est plus restrictive : on a, en plus, $X \not\longrightarrow (X_1, X_2)$ de sorte que le nouveau X_o sera différent du précédent.
Le nouveau consensus (X_1, X_2) impliqué par ce nouvel X_o sera différent du précédent d'après (2) appliqué au nouvel ensemble de X.

- L'ensemble de produits fondamentaux (satisfaisant la condition (3)) étant fini, ainsi que la suite C, on ne pourra indéfiniment ajouter à F un consensus de C, de sorte que X_p appartiendra à une des formes F_i égales à F, c'est-à-dire que X_p appartiendra à C.

Théorème 7.

Si F est la somme des implicants primitifs d'une fonction f, alors nous pouvons obtenir la somme des implicants primitifs de $\bar{f}$ en inversant F et faisant une suppression de multiples sur le résultat.

Théorème 8.

Si F et G sont les sommes des implicants primitifs des fonctions f et g, alors nous pouvons obtenir la somme des implicants primitifs de (f . g) en multipliant F par G et en opérant une suppression de multiples sur le résultat.

2.4. Algorithme de recherche de tous les implicants primitifs d'une forme normale disjonctive F.

Le théorème 6 nous permet de construire un tel algorithme
Il suffit de calculer la suite complète S_c associée à la suite S des X_k.
On a : S_c = (suite d'implicants primitifs) U (suite d'implicants non primitifs).

Si X est un implicant non primitif, il existe X_1, implicant primitif tel que :

$$X \longrightarrow X_1 \longrightarrow F$$

Par conséquent, X est un multiple d'un élément de (suite d'implicants primitifs).

Il suffit donc, pour ne garder que la suite des implicants primitifs, de supprimer dans S_c tous les éléments multiples d'un élément de la suite de S_c : cette opération s'appelle "suppression de multiples".
Il est évident que cette suppression de multiples peut se faire sur toute suite intermédiaire S_i.

D'où l'algorithme suivant :

effectuer dans $S_o = S$, une suppression de multiple
I = 1
calculer S_I; effectuer une suppression de multiple dans S_I
tant que $S_I \neq S_I - 1$
I : = I + 1
calculer S_I
effectuer une suppression de multiple dans S_I

Une autre façon de faire :

Théorème 9.

En inversant deux fois une fonction booléenne et en opérant une suppression de multiples après chaque inversion, on obtient la somme de ses implicants primitifs.

2.5. Recherche des bases irredondantes.

On appelle base d'une fonction booléenne, toute somme d'implicants primitifs égale à la fonction.

La somme de tous les implicants primitifs possède cette propriété : c'est la base complète.

On appelle base irredondante, une base qui cesse d'être base si on enlève un des implicants primitifs qui y figurent.

Par définition de base irredondante et d'implicant primitif, toutes les bases irredondantes de F sont des formes simplifiées de F.

Il s'agit maintenant de trouver toutes les bases irredondantes d'une fonction F, étant donné tous ces implicants primitifs :

$$S = (X_1, X_2, \ldots, X_p)$$

Par exemple : S = (ab', ac, bc, a'bd)

Quelle que soit une base irredondante $\omega = (X_p \ldots X_q \ldots X_r)$

(1). Les éléments X_k de S tels que X_k appartient à S et X_k n'appartient pas à ω sont tels que

$X_k \longrightarrow$ une somme partielle de ω

(puisque dans ce cas, X_k est redondant par rapport à ω)

(2). Les éléments X_q de ω sont tels que

X_q n'implique aucune somme partielle de ω ne contenant pas X_q

(car autrement, X_q serait redondant).

Etant donné $F = \sum_i X_i$, nous définissons la fonction de consensus: (base complète)

$$CONS = \sum_i i \quad X_i$$

(base complète)

<u>dans l'exemple</u>, on a : CONS = 1 ab' + 2 ac + 3 bc + 4 a'bd

On peut calculer la base complète de CONS, à l'aide de l'algorithme du paragraphe 4 :

$$S_{CONS} = (1\ X_1 \ \ldots\ i\ X_i \ \ldots\ p\ X_p, \text{(consensus)})$$

I) Si nous trouvons dans la suite des consensus, un élément de la forme $(ij\ X_k)$, où X_k appartient à S, cela signifie que :

$$ij\ X_k' = (\ i\ X_i, j\ X_j);\ X_k = (X_i, X_j)$$

donc: X_k implique une somme partielle de S :

$$X_k \longrightarrow X_i + X_j$$

D'après les remarques (1) et (2), on peut dire :

si l'on retient dans la base irredondante X_i et X_j, on ne peut retenir X_k.

Par contre, rien n'empêche de retenir (X_k et X_i) ou (X_k et X_j) ou (X_k).

Si nous notons par leur indice (p ... q ... r) la suite des éléments d'une base irredondante, on aura dans cette base :

$$i\ j\ \bar{k}$$

$$\text{ou}\quad k\ i\ \bar{j}$$

$$\text{ou}\quad k\ j\ \bar{i}$$

$$\text{ou}\quad k\ \bar{j}\ \bar{i}$$

c'est-à-dire : $i\ j\ \bar{k} + k\ i\ \bar{j} + k\ \bar{i}\ j + k\ \bar{j}\ \bar{i}$

$$= \quad i\ j\ \bar{k} \;+\; \bar{j}\ k \;+\; \bar{i}\ j\ k$$

$$= \quad i\ j\ \bar{k} \;+\; k\ (\bar{j} + \bar{i})$$

$$= \quad (i\ j) \oplus k$$

II) Si on ne trouve pas dans la suite des consensus, un élément de la forme $i\ j\ X_k'$ (où $k'\ X_k'$ est un élément formant la fonction CONS), on a :

$$X_k' \longrightarrow F = \sum_{i\ i} X$$

comme X_k' n'appartient pas à la suite des consensus de $S' = S \setminus \{ X_k' \}$,

X_k' n'est pas un implicant primitif de $G = \sum_{i \neq k'} X_i$

X_k' ne peut être un implicant non primitif de G car alors il serait un implicant non primitif de $F = G + X_k'$

X_k' n'est donc pas un implicant de G, en conséquence de quoi il n'implique aucune somme partielle de S ne contenant pas X_k'

X_k' doit donc se trouver dans la base irredondante.

En faisant les produits $((ij) \oplus k) \quad k'X \ldots$

et en utilisant les propriétés : $(a \oplus b)\, c = ac \oplus bc$

$$ab \oplus a = a\bar{b}$$

on trouve : $\omega_m \oplus \omega_n \oplus \ldots$

où chaque $\omega = (p \ldots q \ldots r)$ représente une base irredondante de F.

En raisonnant sur l'exemple :

$S = (ab', ac, bc, a'bd)$

$CONS = ab'1 + ac\ 2 + bc\ 3 + a'bd\ 4$

$S_{CONS} = (ab'1;\ ac\ 2;\ ac\ 1\ 3;\ bc\ 3;\ a'bd\ 4;\ bcd\ 2\ 4)$

ac 1 3 apparaît dans la suite des consensus, on garde donc dans la base irredondante :

$$(2 \oplus 1\ 3)$$

le produit donne : $1\ (2 \oplus 1\ 3)\ 3\ 4 = 1\ 2\ 3\ 4 \oplus 1\ 3\ 4$

$$= 1\ \bar{2}\ 3\ 4$$

et la seule base irréductible est :

$$X_1 + X_3 + X_4 = ab' + bc + a'bd$$

Exemple : Calculer les bases irredondantes de la fonction

$$F = ab' + bc' + ca'$$

1°) base complète = $(ab', bc', ca', ac', b'c, ba')$

2°) $CONS = ab'1 + bc'2 + ca'3 + ac'4 + b'c\ 5 + ba'6$

3°) $S_{CONS} = (ab'1, bc'2, ca'3, ac'4, b'c\ 5, ba'6, ac'12, b'c\ 13,$
$ba'23, ab'45, bc'46, ca'56)$

4°) le produit

$$(1 + 45)\ (2 + 46)\ 3 + 56)\ (4 + 12)\ (5 + 13)\ (6 + 23)$$

donne après transformation :

1 3 4 6 + 4 5 2 3 + 4 5 6 + 1 2 3 + 1 2 5 6

D'où les 5 bases irredondantes et les 5 formes simplifiées de F

ac' + b'c + ba'

ab' + bc' + ca'

ab' + ca' + ac' + a'b

bc' + ca' + ac' + b'c

ab' + bc' + b'c + a'b

*

* *

CHAPITRE II.

LOGIQUE DES PROPOSITIONS

Nous entreprenons ici l'étude de deux types particuliers d'énoncés : les propositions simples qui sont constituées d'un sujet, de la copule "être" et d'un prédicat (par exemple : "les roses sont rouges"; le cheval hennit", proposition qui peut se reformuler "le cheval est hennissant") et les propositions composées, constituées de plusieurs propositions simples reliées entre elles par des opérateurs logiques.

Des problèmes sont abordés, tels que le test de la validité d'une conclusion ou la recherche de conclusions à des énoncés donnés, et des méthodes de résolution leur sont apportées.
Certains énoncés y sont cependant rebelles : ce sont les syllogismes classiques, dont une analyse exhaustive clôture ce chapitre.

1. EVALUATION DES PROPOSITIONS COMPOSEES PAR TABLE DE VERITE.

Toute proposition peut être soit vraie, soit fausse; nous dirons qu'elle possède une valeur de vérité : vrai ou faux.
La logique des propositions étudie comment la vérité ou la fausseté des propositions composées est fonction de la vérité ou de la fausseté des propositions simples qui la composent; cette logique ne s'intéresse qu'aux valeurs de vérité des propositions et en aucun cas à leur signification (c'est dans ce sens que nous disons d'une proposition composée qu'elle est une fonction de vérité de ses composants).

A titre d'exemple, l'énoncé

"le sang est rouge ou l'herbe est verte"

n'est pas automatiquement vrai.
La logique des propositions nous autorise à dire qu'il est vrai si l'un ou l'autre des énoncés (ou les deux) "le sang est rouge", "l'herbe est verte" est vrai.

Nous représentons les propositions simples par des variables propositionnelles (une lettre minuscule f, g, ...) et nous notons d'une majuscule F, S ... les énoncés formés à l'aide des variables propositionnelles et d'opérateurs logiques.

1.1. La négation (opérateur logique non).

Quel que soit un énoncé S, le nier, c'est affirmer que sa négation (non S) est vraie.
La négation d'un énoncé sera vraie chaque fois que l'énoncé est faux et sera fausse chaque fois que l'énoncé est vrai.

Nous notons la négation d'un énoncé S par $\bar{S}$ ou S'.

Exemple : si nous admettons que la proposition "l'herbe est rouge" est fausse, nous devons admettre que sa négation "l'herbe n'est pas rouge" est vraie.

propriété : $\bar{\bar{S}} = S$

1.2. La conjonction (opérateur logique et)

La conjonction de deux énoncés quelconques S et F (notée ($S \wedge F$ ou SF) est vraie si, et seulement si, les deux énoncés sont vrais. Elle est fausse si l'un au moins des deux énoncés est faux.

Exemple : si nous tenons pour vrai l'énoncé "le sang est rouge et l'herbe est verte", nous admettons que les deux énoncés "le sang est rouge" et "l'herbe est verte" sont vrais.

propriétés : soient 3 énoncés S, F, U :

$S(FU) = (SF)U = SFU$

$SF = FS$

$SS = S$

1.3. La disjonction (opérateur logique ou).

La disjonction de deux énoncés quelconques F, S (notée S V F) est vraie si l'un des deux est vrai ou si tous les deux sont vrais. Elle est fausse si les deux énoncés sont faux.

Exemple : si nous réfutons l'énoncé "le sang est vert ou l'herbe est rouge", nous soutenons que les deux propositions "le sang est vert", "l'herbe est rouge" sont fausses.

D'autre part, si nous affirmons que l'herbe est verte, nous devons accepter comme vraie la proposition :

" 2 x 2 = 5 ou l'herbe est verte"

Quiconque n'est pas familier au langage de la logique acceptera difficilement la vérité d'une telle proposition. Cela provient de l'usage fait dans la langue courante du mot 'ou'. D'ordinaire, nous n'affirmons la disjonction de deux propositions que si nous croyons que l'une d'elle est vraie, sans savoir laquelle. Si, par exemple, nous regardons un gazon normalement éclairé, il ne nous viendra pas à l'esprit de dire que "l'herbe est rouge ou verte" puisque nous pouvons affirmer quelque chose de plus simple et de plus fort, à savoir que "l'herbe est verte".

Dorénavant, nous attribuerons aux valeurs de vérité (vrai ou faux) d'une variable propositionnelle, les valeurs d'une variable booléenne (respectivement 1 ou 0).

De cette façon, les propositions composées peuvent être représentées par des fonctions de variables booléennes substituées aux variables propositionnelles. Les schémas obtenus par cette substitution sont appelés 'schémas vérifonctionnels'.

Nous pouvons alors évaluer la valeur de vérité d'un énoncé en fonction des valeurs de vérité des propositions qui le composent, au moyen d'une table dite 'de vérité'

Par exemple, pour l'énoncé p V q, nous évaluons la fonction booléenne p + q des 2 variables p et q

p	q	p + q
1	0	1
0	0	0
1	1	1
0	1	1

et nous dirons que p V q est vrai ssi p + q = 1
p V q est faux ssi p + q = 0

1.4. La disjonction exclusive (opérateur logique ou bien ... ou bien).

La disjonction exclusive de deux énoncés S, F (notée S ⊕ F) est vraie uniquement si l'un des deux est vrai, à l'exclusion de l'autre.

Exemple : si nous supposons vrai le fait que Paul est parti se distraire au cinéma (et sachant qu'il n'existe que deux salles en ville : le Caméo et l'Eldorado), nous pouvons dire : ou bien Paul est au Caméo (p), ou bien il est à l'Eldorado (q)" et écrire la table de vérité :

p	q	p ⊕ q
1	0	1
0	0	0
1	1	0
0	1	1

p ⊕ q est faux si p et q sont égales à 1 (Paul ne peut pas être à la fois au Caméo et à l'Eldorado);

p ⊕ q est également faux si p = 0 et q = 0 (nous savons que Paul est dans l'une des deux salles).

1.5. Le conditionnel (opérateur logique si ... alors ...).

Soient deux énoncés quelconques S et F, le conditionnel "si S alors F" (noté S ⟩ F) est une proposition fausse uniquement si S est vrai et F est faux.

S est l'antécédent du conditionnel; F est son conséquent.
Nous pouvons écrire la table de vérité (nous avons déjà vu au chapitre I qu'un conditionnel S ⟩ F peut se réécrire $\bar{S}$ + F

S	F	$\bar{S}$ + F
0	0	1
0	1	1
1	0	0
1	1	1

Remarquons que si l'antécédent est faux et quelle que soit la valeur de vérité du conséquent, le conditionnel S ⟩ F est vrai.

Exemple : "si tu résous ce problème, alors je mange mon chapeau"
Nous pouvons écrire, en remplacant l'antécédent par p et le conséquent par q :

$$p' + q = 1$$

En fait, en soutenant ce conditionnel, notre intention est d'affirmer la fausseté du conséquent et donc, la fausseté de l'antécédent ("tu ne peux résoudre ce problème").

Certaines propositions logiquement vraies peuvent paraître choquantes dans le langage naturel.

Par exemple : les 3 propositions suivantes sont logiquement vraies :

"Si la France est en Europe, alors la mer est salée".
"Si la France est en Amérique, alors la mer est salée".
" Si la France est en Amérique, alors la mer est douce".

Dans le langage courant, ces propositions sont difficilement perçues comme ayant une signification, et moins encore comme étant vraies.
La difficulté provient du fait qu'il n'est guère habituel de former de telles propositions, alors que la valeur de vérité des énoncés composants est déjà connue inconditionnellement : pourquoi former ces conditionnels alors que nous sommes en mesure d'affirmer "la mer est salée" et de nier "la France est en Amérique" ?

Dans la pratique, celui qui déclare "si p alors q" ne connaît pas les valeurs de vérité des propositions p et q, mais il entend affirmer la fausseté de (p et (non q)).

Nous disons :

"Si Paul a la malaria, alors il a besoin de quinine"

parce que nous savons qu'il est impossible d'avoir la malaria sans avoir besoin de quinine, mais que nous sommes dans le doute sur la maladie de Paul et sur son besoin de quinine.

1.6. L'homologie.

Nous avons déjà vu au premier chapitre que l'homologie $S \equiv F$ (ou $\bar{S}\,\bar{F} + S\,F$) est fausse uniquement si

S est vrai et F est faux
S est faux et S est vrai,

d'où la table de vérité suivante :

S	F	$S \equiv F$
0	0	1
0	1	0
1	0	0
1	1	1

2. LES TAUTOLOGIES.

Il est évident que l'énoncé :

"le cheval hennit ou le cheval ne hennit pas"

est vrai.

Mais bien plus l'énoncé

p ou (non p)

est toujours vrai, pour toutes les propositions que l'on substitue à p.
Il en est de même de l'énoncé "si p alors (p ou q)".
Nous dirons de tels énoncés, où seule compte la structure logique (... ou non ..., si ... alors ... ou ...) qu'ils sont valides et nous parlerons, pour les désigner, de "tautologies" ou de "lois de la logique des propositions".
Pour tester la validité d'un énoncé (c'est-à-dire décider s'il est valide ou non), il nous suffit d'écrire sa table de vérité et d'observer la dernière colonne : celle-ci contient uniquement des '1' si, et seulement si, l'énoncé est valide.

Exemple :

p	p + p'
1	1
0	1

Il est évidemment essentiel de bien distinguer les tautologies des énoncés simplement vrais, mais dont la valeur de vérité dépend de la vérité de ses composants.

2.1. L'implication.

Nous dirons qu'un énoncé F implique un énoncé S (et nous noterons F $\longrightarrow$ S) si et seulement si :

lorsque F est vrai, nous pouvons en déduire que S est vrai
lorsque F est faux, nous ne pouvons rien déduire de S.

Autrement dit, $F \longrightarrow S$ si et seulement si

$$\bar{F} + S = 1$$

quelles que soient les valeurs de vérité des variables qui composent F et S.

Nous voyons que l'implication $F \longrightarrow S$ est un conditionnel $F \supset S$ VALIDE.
Il est donc toujours possible de décider si un énoncé F implique un énoncé S.
Il suffit de former le conditionnel $F \supset S$ et de vérifier sa validité au moyen d'une table de vérité.

Par exemples : (1) $q' \longrightarrow (pq)'$
puisque $q + p' + q' = 1$

(2) $p \longrightarrow (q \longrightarrow p)$
car $p' + q' + p = 1$

(3) $(p \longrightarrow q) \longrightarrow (q' \longrightarrow p')$
car $pq' + q + p' = q' + q + p = 1$

(4) $((p \longrightarrow q) \wedge (q \longrightarrow r)) \longrightarrow (q \longrightarrow r)$
car $pq' + qr' + q' + r = q + q' + r = 1$;

cette dernière implication est ordinairement connue sous le nom de 'propriété de transitivité.

2.2. L'équivalence.

De la même façon qu'une implication est la validité d'un conditionnel, une équivalence est la validité d'une homologie.
Par définition, deux énoncés F et S sont équivalents (nous noterons $F \longleftrightarrow S$) si et seulement si l'homologie ($F \equiv S$) est valide.
Cela signifie que deux énoncés sont équivalents s'ils concordent quant à leur valeur de vérité pour toute interprétation de leurs variables propositionnelles.

Par exemple : $p\,(q \vee r) \longleftrightarrow pq \vee pr$

car $\overline{p\,(q + r)}\;\overline{(pq + pr)} + p(q + r)(pq + pr)$

$= (p' + q'r')(p' + q')(p' + r') + pq + pr$

$= p' + q'r' + pq + pr$

$= p' + r' + q + r$

$= 1$

2.3. Les axiomes.

Certaines tautologies sont parfois utilisées comme axiomes. Associées à des règles d'inférence, elles permettent d'engendrer tous les théorèmes (= énoncés valides) de la logique des propositions.
L'une de ces règles est le modus ponens:

> si un théorème ou un axiome est un conditionnel dont l'antécédent est un théorème,alors le conséquent est un théorème.

Une autre règle est la substitution :

> on peut substituer n'importe quel schéma à toutes les occurrences d'une lettre, dans un théorème et obtenir ainsi un nouveau théorème.

Les axiomes habituellement employés sont indépendants : aucun d'eux n'est dérivable des autres comme théorème.
Mais l'intérêt d'un système d'axiomes dans la logique des propositions est douteux : rappelons, en effet, que nous disposons d'une procédure de décision (les tables de vérité) nous autorisant à tester la validité de tout schéma vérifonctionnel. En revanche, l'échec à démontrer la validité d'un schéma par la méthode axiomatique peut signifier soit la non validité, soit l'absence de chance dans notre utilisation des axiomes et règles d'inférence.

3. APPLICATIONS.

3.1. Tests de validité.

Nous avons vu qu'au départ d'énoncés, on pouvait opérer une transposition en fonctions booléennes.
Nous avons également vu, à propos des tautologies, qu'un énoncé est valide s'il est vrai pour toutes les interprétations des variables propositionnelles qui le composent (ou s'il vaut 1 pour toutes les interprétations des variables booléennes qui le composent).
Nous savons, enfin, qu'il est toujours possible de tester la validité d'un énoncé de la logique des propositions. Le moyen utilisé pour ce faire est la table de vérité, combiné à l'application des diverses opérations de simplification booléenne (opérations basées sur les relations générales vues au chapitre I).

Les exercices qui suivent consistent tous à tester la validité d'une conclusion C pour des énoncés formés de prémisses P_i.

$$P_1\ P_2 \ldots P_n \supset C$$

La conclusion est valide si elle peut être considérée comme vraie partout où S l'est, c'est-à-dire :

$$P_1\ P_2 \ldots P_n \longrightarrow C$$

$$\overline{P_1\ P_2 \ldots P_n} + C = 1$$

ou $\quad P_1\ P_2 \ldots P_n\ \bar{C} = 0$

---

1. P_1 : Si le budget n'est pas amputé (b') alors une condition nécessaire et suffisante pour que les prix restent stables (p) est que les taxes seront accrues (t).

P_2 : Les taxes augmenteront (t) si et seulement si le budget n'est pas réduit (b').

P_3 : Si les prix restent stables (p) alors les taxes ne seront pas augmentées (t').

C : Les taxes ne seront pas augmentées (t')

P_1 : $b' \supset (p \equiv t)$ ou $b + pt + p't'$

P_2 : $t = b'$ ou $tb' + t'b$

P_3 : $p \supset t'$ ou $p' + t'$

b	p	t	P_1	P_2	P_3	πP	C	$\pi P \supset C$
0	0	0	1	0	1	0	1	1
0	0	1	0	1	1	0	0	1
0	1	0	0	0	1	0	1	1
0	1	1	1	1	0	0	0	1
1	0	0	1	1	1	1	1	1
1	0	1	1	0	1	0	0	1
1	1	0	1	1	1	1	1	1
1	1	1	1	0	0	0	0	1

conclusion valide

2. Si la France est une démocratie (p), ses citoyens ont le droit de vote (q).
Les citoyens ont le droit de vote (q).

La France est une démocratie (p).

$$(p' + q)\ q\ p' = p'q \neq 0$$

conclusion non valide

3. Dans une démocratie (p), le chef du gouvernement est élu au suffrage universel (q).
En France, le chef du gouvernement n'est pas élu au suffrage universel (q').

La France n'est pas une démocratie (p').

$$(p' + q)\ q'p = p'q'p + q\ q'p = 0$$

conclusion valide

4. Si un homme est communiste (p) alors il critique la police du gouvernement (q).
Cet homme critique la police du gouvernement (q).

Cet homme est communiste (p).

$$(p' + q)\ q\ p' = p'q \neq 0$$

conclusion non valide

5. Si les jalousies sont baissées (p), c'est qu'il n'y a personne à la maison (q').
Les jalousies sont levées (p').

Il y a quelqu'un (q)

$$(p' + q')\ p'\ q' \neq 0$$

conclusion non valide

6. Si tu n'étudies pas (p'), tu ne réussiras pas (q').
Si tu étudies (p), tu seras récompensé (r).

Si tu réussis (q), tu seras récompensé (r).

$$(p + q')(p' + r)\ qr' = (pr + q'r + q'p')\ qr' = 0$$

conclusion valide

7. Leibniz (p) ou Newton (q) a découvert la gravitation.
Newton l'a découverte (q).

Leibniz ne l'a pas découverte (p').

$$(pq' + p'q)\ qp = 0$$

conclusion valide

8. Si Landru est un assassin (p), il a tué ses amies (q) et les a incinérées (r).
Il n'a pas tué ses amies (q').

Landru n'est pas un assassin (p').

$$(p' + qr)\, q'p = 0$$

conclusion valide

9. S'il veut réussir (p), il doit être à la fois compétent (q) et chanceux (r).
Car s'il est incompétent (q'), il lui est impossible de réussir (p').

S'il n'a pas de chance (r'), il est certain que quelque chose ira mal (p').

$$(p' + qr)(q + p')r'p = (p' + qr)\, r'p = 0$$

conclusion valide

10. Pour que le candidat soit accepté (p), il suffit qu'il emporte la décision du chef (q).
Il n'emportera cette décision que s'il est bien habillé (r).
Il ne s'habillera pas bien.

Il ne sera pas accepté (p').

$$(q' + p)(r' + q)\, r'p = r'\, p \neq 0$$

conclusion non valide

11. Si août est pluvieux (p), alors les fleurs s'épanouissent en mai (q) et les moustiques volent en juin (r).
Si les moustiques volent en juin (r) alors les maladies augmentent durant les vacances (s).
Si les fleurs s'épanouissent en mai (q) alors il y aura beaucoup de miel en septembre (t).
Si août n'est pas pluvieux (p') alors les pelouses seront brunes en été (u).

Ainsi, il y aura beaucoup de miel en septembre (t) et les maladies augmenteront en vacances (s) ou les pelouses bruniront en été (u).

$$\overline{(p' + qr)(r' + s)(q' + t)(p + u) + ts + u}$$

$$pq' + pr' + rs' + qt' + p'u' + ts + u$$

$$q' + r' + s' + t' + p' + s + u = 1$$

conclusion valide

3.2. Recherche d'une conclusion.

Le principe de résolution des problèmes proposés est le suivant : étant donné un ensemble de propositions $(a_1, a_2, \ldots, a_n)$ supposées vraies, nous pouvons faire la conjonction :

$$\prod_{i=1}^{n} a_i = P$$

et conclure que la proposition P sera également vraie.
En substituant des variables booléennes aux propositions, P devient une fonction booléenne. En simplifiant cette fonction, nous obtenons une proposition vraie qui n'était peut-être pas directement lisible dans l'ensemble $(a_1, a_2, \ldots, a_n)$ initial.

Par exemple : imaginons les deux prémisses :

$$(1) \quad p \longrightarrow q$$

$$(2) \quad q \longrightarrow r$$

par Transitivité

Nous pouvons en conclure que $p \longrightarrow r$

- soit grâce à la propriété de transitivité
- soit au prix d'un petit effort supplémentaire, en faisant le produit de (1) avec (2)

$$P = (p' + q)(q' + r) = 1$$

$$P = p'q' + qr + p'r = 1$$

après simplification :

$$P = p'q' + qr = 1$$

autrement dit :

$$\bar{P} = 0$$

$$pq' + pr' + qr' = 0$$

ceci est vrai si $pq' = 0$ (a)

et $pr' = 0$ (b)

et $qr' = 0$ (c)

(b) signifie $p' + r = 1$, d'où la conclusion C :

$$p \longrightarrow r$$

Notons que P est une conclusion dans le sens où

$\prod a_i$ implique P

(puisque $P \longrightarrow P$)

mais observons que les transformations subies par P nous font passer

d'une égalité $\prod a_i = P$

à une implication $\prod a_i \longrightarrow C$

(nous n'avons plus $\prod a_i = C$ puisque $(p' + q)(q' + r) \neq (p' + r)$

1. Une énorme quantité de marchandise a été dérobée dans un magasin. Le voleur (ou les voleurs) s'est enfui en voiture. Trois malfaiteurs notoires A, B, C sont convoqués à Scotland Yard pour y être interrogés. On peut établir les faits suivants de façon certaine :

(1) Nul autre que A, B, C n'a pu participer au cambriolage.

(2) C ne fait jamais un coup sans la complicité de A.

(3) B ne sait pas conduire.

En notant a : ' A est coupable '

b : ' B est coupable '

c : ' C est coupable '

le produit des prémisses donne :

$$(a + b + c)\,(c' + a)\,(b' + a + c) = a$$

d'où la conclusion :

' A est coupable '

2. Voici une autre affaire de vol : quand A, B et C furent conduits à Scotland Yard pour y être interrogés, les faits suivants furent établis de façon certaine :

(1) Nul autre que A, B, C ne pouvait être impliqué dans l'affaire.

(2) A ne travaillait jamais sans au moins un complice.

(3) C était innocent.

le produit de prémisses :

$$(a + b + c)\,(a' + b + c)\,c' = (b + c)\,c' = bc'$$

d'où la conclusion :

' B est coupable '

3. Voici le cas plus intéressant d'un cambriolage commis à Londres, pour lequel trois malfaiteurs notoires, A, B et C furent interpelés et entendus à Scotland Yard. Détail très important : A et C étaient des vrais jumeaux et ils se ressemblaient tant qu'à peu près personne ne pouvait les distinguer. Ces trois malfaiteurs avaient un dossier très chargé et leurs habitudes étaient bien connues. En particulier, on savait que les jumeaux n'étaient pas téméraires, au point qu'aucun d'eux n'aurait osé entreprendre un coup sans complice. Au contraire, B aimait agir seul et il ne s'encombrait jamais de complice. D'autre part, plusieurs témoins avaient affirmé qu'à l'heure du vol, ils avaient vu l'un des jumeaux dans un bar de Douvres, mais sans pouvoir dire lequel c'était.

En supposant cette fois encore que nul autre que A, B ou C ne pouvait être impliqué dans ce vol, qui était le coupable, qui était innocent ?

Nous discernons les prémisses :

$$a \supset c + b$$

$$c \supset a + b$$

$$b \supset a'c'$$

$$a' + c'$$

$$a + b + c$$

dont le produit donne :

$$(a' + c + b)(c' + a + b)(b' + a'c')(a' + c')(a + b + c)$$

$$= a'\, c'\, b$$

dont la conclusion :

B est coupable

4. Il y a 3 personnes Albert (a), Bernard (b) et Charles (c); ils peuvent avoir les cheveux blonds ou bruns. C'est à nous de le déterminer étant donné que :

1. A moins que c n'ait les cheveux blonds, b a les cheveux de couleur différente de ceux d'Albert.

2. Si a est blond, b à la même couleur de cheveux que c.

3. Si b et c sont tous deux bruns, a l'est également.

4. Si a est brun et b blond, alors c est brun.

5. Si b est blond et seulement s'il l'est, a et c sont blonds.

Les symboles a, b, c sont considérés comme des variables binaires s'écrivant sous cette forme ou sous la forme inverse (a', b', c') suivant qu'elles se rapportent à une personne blonde ou brune :

$$c' \supset ba' + b'a$$

$$a \supset bc + b'c'$$

$$b'c' \supset a'$$

$$a'b \supset c'$$

$$b \equiv ac$$

Le produit de ces prémisses fournit :

$$a\ b\ c$$

d'où les conclusions :

A, B et C sont blonds.

5. Monsieur et Madame Jones désirent acheter une voiture.
Tous les modèles peuvent être fournis avec transmission automatique (a), toit ouvrant (b), ceintures de sécurité (c), auto-radio (d) ou sièges escamotables (e).

· Monsieur Jones déclare :

" Si nous prenons la transmission automatique, il nous faut un toit ouvrant et des ceintures de sécurité".

Son épouse réplique :

" S'il faut un toit ouvrant, alors je veux une auto-radio et des sièges escamotables".

Et Monsieur Jones de répondre :

" Si nous prenons le toit ouvrant et l'auto-radio, nos moyens ne nous permettent pas de prendre une transmission automatique".

Les prémisses traduites :

$$a \supset bc$$
$$b \supset de$$
$$bd \supset \bar{a}$$

leur produit donne :

$$a'b' + a' d e = a' (b' + d e)$$

d'où la conclusion :

"Jones ne prendra pas une transmission automatique".

6. Vous désirez acheter une automobile "personnalisée", dans la gamme de celles que vous offre la marque "X".
Repoussant d'emblée les véhicules du type touriste standard, vous demandez au vendeur de vous préciser les différents aménagements ou accessoires que vous pouvez obtenir; ce sont les suivants :

- cylindrée : faible (w_0), moyenne (w_1), forte (w_2)

- carrosserie : carrosserie italienne (x), carrosserie en plastique (x')

- embrayage, boîte de vitesses; il existe deux modèles : embrayage automatique à 3 vitesses (y) ou embrayage ordinaire, mais boîte à 4 vitesses (y')

- toit : véhicule à toit ouvrant (z) ou cabriolet décapotable (z').

Votre choix se limitera à ces possibilités. Le vendeur vous précise en outre que :

1.- Si vous choisissez le cabriolet décapotable, vous ne pouvez choisir ni la carrosserie en plastique, ni une faible cylindrée.

2.- La boîte à 4 vitesses est incompatible avec une faible cylindrée.

3.- La carrosserie en plastique est incompatible avec une forte cylindrée.

4.- Un cabriolet décapotable implique soit une boîte à 4 vitesses, soit une cylindrée forte ou moyenne.

Pour fixer votre choix (la couleur ne faisant aucune difficulté), vous décidez de vous en remettre au prix et de choisir la voiture personnalisée qui vous reviendra le moins cher. Le vendeur vous tend le barème réduit suivant : (prix en sus du modèle standard) :

w_1 = 250 Frs et w_2 = 500 Frs; x = 500 Frs et x' = 300 Frs; y = 400 Frs, y' = 280 Frs; z = 150 Frs et z' = 200 Frs.

Quelle voiture choisirez-vous ?

Les lettres w avec indice, ainsi que les lettres x, y, z et leurs compléments ont été introduites dans le texte de l'énoncé du problème. Chacune de ces lettres correspond à une proposition bien définie et sa signification logique est évidente lorsque l'accessoire auquel elle se rapporte est de nature bivalente.

Par contre, la proposition concernant la cylindrée est de nature trivalente. Afin d'éviter l'introduction d'une notion nouvelle, nous caractériserons la cylindrée faible, moyenne ou forte respectivement par la variable binaire w_0, w_1 ou w_2 égale à 1.

Dans ces conditions, par exemple, la relation $w_0' = w_1 \oplus w_2 = w_1 w_2' + w_1' w_2$ ou même $w_1 + w_2$ est évidente car si l'on ne choisit pas la cylindrée faible, on ne peut adopter que la cylindrée moyenne en excluant la forte ou la cylindrée forte en excluant la moyenne. En raisonnant très simplement, on remarque d'autre part qu'un produit logique tel que $w_1 w_2$ est nul car on ne peut simultanément adopter les cylindrées moyenne et forte tandis que $w_1' w_2' = (w_0 + w_2)(w_0 + w_1) = w_0$

Les quatre points proposés dans l'énoncé du problème se traduiront par :

1. " S z' alors non $(x' + w_0)$" est un conditionnel qui s'exprime par :

$$z + \overline{(x + w_0)} = z + xw_0' \qquad (1)$$

2. "y' et w_0 s'excluent" d'où : $yw_0 + y'w_0'$ (2)

3. "x' et w_2 s'excluent" d'où : $xw_2 + x'w_2'$ (3)

4. "z' implique soit y', soit $(w_1 \oplus w_2)$" d'où :

$$z + \left[y'\overline{(w_1w_2' + w_1'w_2)} + y(w_1w_2' + w_1'w_2) \right] = z + y'w_0 + y(w_1 + w_2) \qquad (4)$$

Si l'on effectue les produits des relations (1).(4) et (2).(3), on trouve :

$$(1) \cdot (4) = z + xy \ (w_1 + w_2) \qquad (5)$$

$$(2) \cdot (3) = xy'w_2 + x'yw_o + x'y'w_1 \qquad (6)$$

Le produit des relations (5) et (6) fournit finalement la solution du problème :

$$xy'zw_2 + x'yzw_o + x'y'zw_1$$

Le point de vue financier des 3 solutions possibles est examiné en remplaçant les variables par leur valeur en francs d'où l'on conclura qu'il s'agira de choisir ($x'yzw_o$), la voiture dont la cylindrée est faible avec carrosserie en plastique et embrayage automatique; le toit de cette voiture étant un toit ouvrant.

7. Pour une jeune fille, le mari idéal devrait être de grande taille (t), bronzé (b), élégant (h).
Elle connaît quatre jeunes gens : Alac (A), Bill (B), Carl (C) et Dave (D).
Un seul de ceux-ci possède les caractéristiques que la jeune fille exige.

1. Trois des jeunes gens sont t; deux sont b, et un seul est h.

2. Chacun des quatre a au moins une des qualités exigées.

3. A et B ont le même teint.

4. B et C ont la même taille.

5. C et D ne sont pas de la même taille.

En convenant des symboles suivants :

A_t : 'Alec est de grande taille'

$\overline{A_t}$: 'Alec est de petite taille'

A_h : 'Alec est élégant'

$\overline{A_h}$: 'Alec n'est pas élégant', etc ...

nous pouvons traduire les faits 1 à 5 :

1. (a) $A_t B_t C_t \bar{D}_t + A_t B_t \bar{C}_t D_t + A_t \bar{B}_t C_t D_t + \bar{A}_t B_t C_t D_t$

   (b) $A_b B_b \bar{C}_b \bar{D}_b + A_b \bar{B}_b C_b \bar{D}_b + A_b \bar{B}_b \bar{C}_b D_b + \bar{A}_b B_b C_b \bar{D}_b$

   $+ \bar{A}_b B_b \bar{C}_b D_b + \bar{A}_b \bar{B}_b C_b D_b$

   (c) $A_h \bar{B}_h \bar{C}_h \bar{D}_h + \bar{A}_h B_h \bar{C}_h \bar{D}_h + \bar{A}_h \bar{B}_h C_h \bar{D}_h + \bar{A}_h \bar{B}_h \bar{C}_h D_h$

2. (a) $A_t + A_h + A_b$

   (b) $B_t + B_h + B_b$

   (c) $C_t + C_h + C_b$

   (d) $D_t + D_h + D_b$

3. $A_b\ B_b + \bar{A}_b\ \bar{B}_b$

4. $B_t\ C_t + \bar{B}_t\ \bar{C}_t$

5. $C_t\ \bar{D}_t + \bar{C}_t\ D_t$

Le produit de 4 et 5 avec 1 (a) donne :

$$A_t\ B_t\ C_t\ \bar{D}_t \qquad (6)$$

Le produit de 3 avec 1 (b) donne :

$$A_b\ B_b\ \bar{C}_b\ \bar{D}_b + \bar{A}_b\ \bar{B}_b\ C_b\ D_b \qquad (7)$$

Le produit de (6) avec (7) donne :

$$A_t\ A_b\ B_t\ B_b\ C_t\ \bar{C}_b\ \bar{D}_t\ \bar{D}_b +$$

$$A_t\ \bar{A}_b\ B_t\ \bar{B}_b\ C_t\ C_b\ \bar{D}_t\ D_b$$

Ce résultat multiplié à 1 (c) et 2 fournit (sachant qu'une seule personne réunit toutes les qualités) :

$$A_t\ \bar{A}_b\ \bar{A}_h\ B_t\ \bar{B}_b\ \bar{B}_h\ C_t\ C_b\ C_h\ \bar{D}_t\ D_b\ \bar{D}_h$$

d'où la conclusion :

"A est t, B est t, C est t, b, h et D est b"

4. LIMITE DE LA LOGIQUE DES PROPOSITIONS.

Imaginons les 3 prémisses :

(a) tous les hommes sont mortels

(b) quelques hommes ne sont pas des chiens

(c) aucun chien n'est mortel

Essayons d'en faire une transposition en fonctions booléennes; nous sommes tentés d'écrire :

(1) si untel est un homme (h), alors untel est mortel (m)

(2) untel est un homme et untel n'est pas un chien (c')

(3) si untel est un chien, alors untel n'est pas mortel

ou encore :

(1) $h' + m$

(2) $h\ c'$

(3) $c' + m'$

Dans ces conditions, nous sommes obligés d'accepter (3) comme conclusion de (1) et (2), puisque :

$$\overline{(h' + m)\ h\ c'} + c' + m'$$

$$h\ m' + h' + c' + m'$$

$$h + h' + c' + m' = 1$$

Or, cette conclusion, "aucun chien n'est mortel" est manifestement erronée. Notre erreur provient naturellement de la traduction de (a) et (c) ou (1) et (3). Ces deux dernières nous parlent d'un seul objet 'untel' (c'est la même qui est évoqué dans la prémisse (2)).

Les prémisses (a) et (c) sont bien plus générales. On voit mal, hélas, comment les rendre dans le formalisme développé jusqu'ici.

A. Syllogismes
B. Sorites - p. 71

5. ANALYSE DES SYLLOGISMES.

Nous connaissons tous le syllogisme classique :

Tous les hommes sont mortels;
or, Socrate est un homme; (1)
donc, Socrate est mortel.

Il est composé de trois prémisses dont la dernière est la conclusion. Chaque prémisse est constituée d'un sujet et d'un prédicat, que nous appellerons 'termes'.

Mais la formulation de ce syllogisme, telle qu'Aristote l'avait donnée est la suivante :

Si tous les hommes sont mortels,
et si Socrate est un homme,
alors Socrate est mortel.

La différence essentielle avec (1) réside en ce que la forme d'Aristote est une proposition, qui est nécessairement vraie ou fausse, alors que la première forme est celle d'une inférence.

La logique doit pouvoir s'appliquer aux hommes comme à toute autre classe d'objets.
Aussi, afin de conserver au syllogisme uniquement sa forme, Aristote avait déjà substitué des lettres aux termes :

Si tout H est M
et si tout G est H (2)
alors tout G est M

Si la proposition (2) est valide (c'est-à-dire si le conditionnel 'si ... si ... alors' est une implication), alors la forme traditionnelle (1) est également valide.

Un syllogisme sera donc valide si sa formulation est telle qu'elle ne peut conduire à partir de prémisses vraies, à une conclusion fausse.

Aristote avait classé les prémisses des syllogismes classiques en quatre formes différentes :

la forme A : 'tout F est H';

la forme E : 'aucun F n'est H';

la forme I : 'quelques F sont H';

la forme O : 'quelques F ne sont pas H'

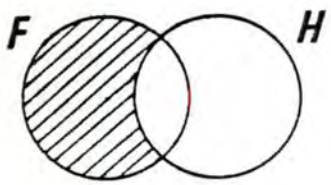
(les 4 voyelles proviennent des mots latins AffIrmo et nEgO).

5.1. Représentation graphique des syllogismes.

Nous développons ici une représentation par diagrammes de Venn, adaptée à l'étude des inférences portant sur trois termes, et en particulier à l'analyse des syllogismes classiques.

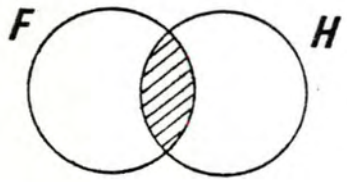
5.1.1. Représentation des quatre formes A E I O.

a) 'tout F est H'



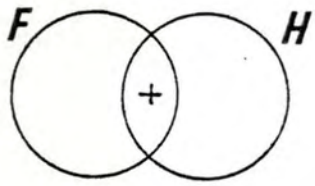
la région hachurée montre qu'il n'existe pas de F qui ne soit pas H.

b) 'Aucun F n'est H'



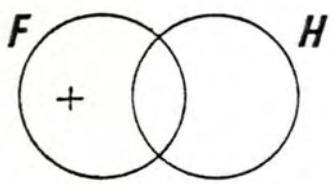
la région hachurée montre qu'il n'existe pas de F qui soit également H

c) 'Quelques F sont H'



la croix montre qu'il existe au moins un F qui est H

d) 'Quelques F ne sont pas H'



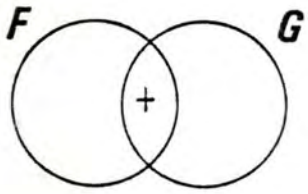
la croix montre qu'il existe au moins un F qui n'est pas H

La blancheur d'une région indique le manque d'informations concernant cette région.
Alors que les hachures signifient le vide, l'absence de hachures ne signifie pas le non vide : le non vide est représenté par une croix.
Par exemple, en remplacant F par 'romancier' et H par 'cinéaste', la phrase 'tout H est F' ne nous donne aucune information sur l'existence des romanciers cinéastes, mais elle nous dit que s'il existe des cinéastes, ceux-ci sont tous des romanciers.

Toujours à titre d'exemples :

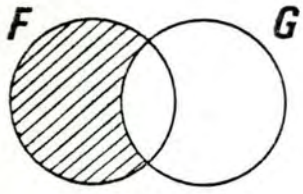
l'inférence 'quelques F sont G
donc il y a des G'

est valide puisque le conditionnel correspondant est toujours vrai, quelle que soit la signification de F et G.



par contre, l'inférence 'tous les F sont G
donc il y a des G'

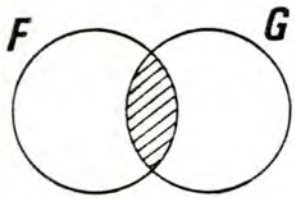
n'est pas valide puisque le conditionnel (tous les F sont G ⊃ il y a des G) n'est pas vraie s'il n'existe pas de F.



5.1.2. Propriétés des quatre formes A E I O.

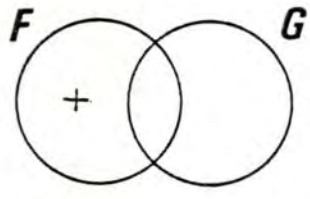
Les diagrammes permettent de visualiser aisément les propriétés suivantes :

a. passage d'une forme à l'autre :



la forme E 'aucun F n'est G'
équivaut à
la forme A 'tout F est non G'

la forme O 'quelques F ne sont pas G'
équivaut à
la forme I 'quelques F sont non G'



b. inversion des termes :

'Tout F est G' équivaut à 'Tout non G est non F'
'Aucun F n'est G' équivaut à 'Aucun G n'est F'
'Quelques F sont G' équivaut à 'Quelques G sont F'
'Quelques F ne sont pas G' équivaut à 'Quelques non G sont F'

5.1.3. Les quatre figures des syllogismes.

La classification des syllogismes en figures a un intérêt historique plutôt qu'une importance théorique. Son but est pratique : elle permet de distinguer dans une application, les syllogismes valides des syllogismes non valides.

Les deux premières prémisses d'un syllogisme classique contiennent toujours un terme commun : le moyen terme T.
L'autre terme de la première prémisse est le grand terme G; l'autre terme de la deuxième prémisse est le petit terme P.

La figure d'un syllogisme dépend de la composition des termes dans son expression :

	figure 1	figure 2	figure 3	figure 4
premisse 1	*TG*	*GT*	*TG*	*GT*
premisse 2	*PT*	*PT*	*TP*	*TP*
conclusion	*PG*	*PG*	*PG*	*PG*

Ce tableau donne l'ordre d'apparition de termes dans chaque prémisse, pour chacune des figures.

Si l'on effectue toutes les combinaisons des quatre prémisses :

AA (A,E,I,O)	AE (A,E,I,O)	AI (A,E,I,O)	AO (A,E,I,O)
EA ( )	EE ( )	EI ( )	EO ( )
IA ( )	IE ( )	II ( )	IO ( )
OA ( )	OE ( )	OI ( )	OO ( )

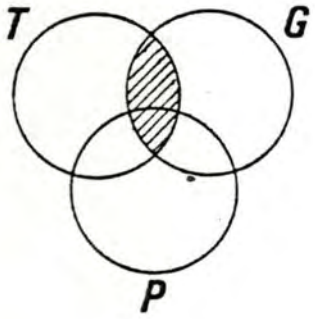
il devrait exister 64 syllogismes. Si, pour chaque syllogisme, on tient compte de 4 figures différentes, il semblerait que 256 syllogismes soient possibles. Beaucoup d'entre eux, cependant, ne sont pas valides.

5.1.4. Tests de validité.

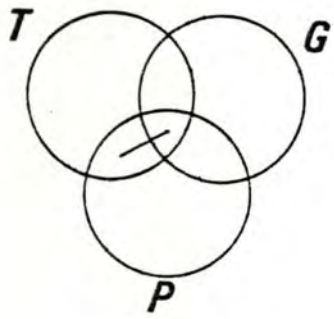
Les diagrammes de Venn permettent de tester facilement la validité d'un syllogisme.
Prenons, par exemple, le syllogisme :

Aucun philosophe n'est méchant	si aucun T n'est G
quelques grecs sont philosophes	et si quelques P sont T
donc quelques grecs ne sont pas méchants	alors quelques P ne sont pas G

la première prémisse nous dit :

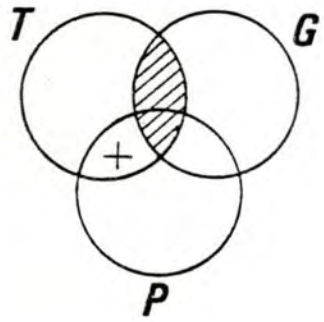


la seconde nous donne :



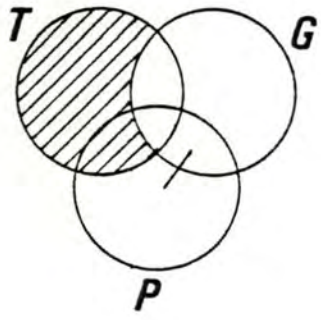
l'innovation qui consiste à remplacer la croix par une barre permet de traverser une limite pour montrer qu'une partie composée n'est pas vide.

les deux prémisses ensemble, nous obtenons :



et nous voyons immédiatement que la conclusion O est vraie.
Examinons à présent le syllogisme AO, 1ère figure

tout T est G;
quelques P ne sont pas T



Une conclusion valide serait 'Il existe des P', mais il n'y a aucune conclusion de la forme A, E, I ou O qui fasse de AO 1ère figure un syllogisme valide.

En utilisant ainsi les diagrammes de Venn, nous pouvons trouver tous les syllogismes valides :

1ère figure	2e figure	3e figure	4e figure
A A A	E A E	I A I	A E E
E A E	A E E	A I I	I A I
A I I	E I O	O A O	E I O
E I O	A O O	E I O	

Neuf autres combinaisons méritent une attention particulière : il s'agit de syllogismes valides si l'on ajoute une condition aux deux premières prémisses.

1ère figure	2e figure	3e figure	4e figure	prémisse ajoutée
AAI, EAO	AEO, EAO		AEO	'il y a des P'
		AAI, EAO	EAO	'il y a des T'
			AAI	'il y a des G'

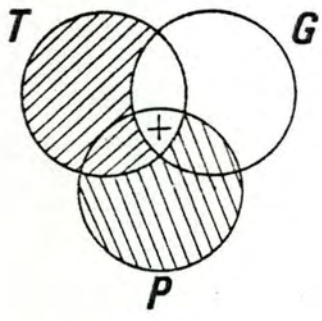
par exemple : A A I 1è figure et 'il y a des P'

les trois conditions si tout T est G

si tout P est T

et si il y a des P

donnent :

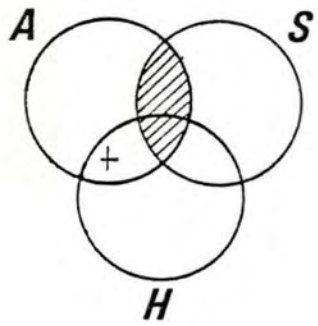


La conclusion I est bien vérifiée.

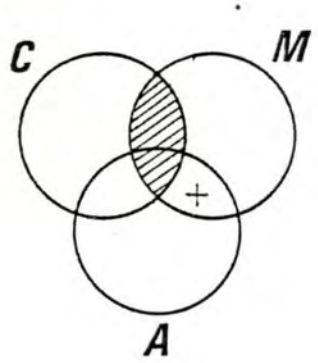
5.1.5. Quelques exemples.

A titre d'exemple, voici quatre syllogismes E I O (1e, 2e, 3e et 4e figures) :

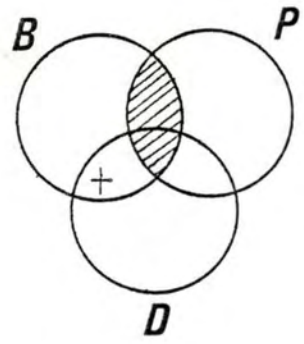
aucun académicien (A) n'est stupide (S);
quelques hommes (H) sont des académiciens;
donc, quelques hommes ne sont pas stupides.



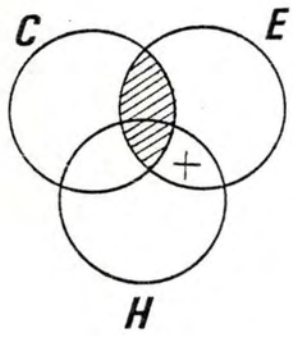
aucun canard (C) n'est mammifère (M);
quelques animaux (A) sont des mammifères;
donc, quelques animaux ne sont pas des canards.



aucun bienfait (B) n'est perdu (P);
quelques bienfaits sont désintéressés (D);
donc, quelques actes désintéressés ne sont pas perdus.



aucune chèvre (C) n'est équidée (E);
quelques équidés sont herbivores (H);
donc, quelques herbivores ne sont pas des chèvres.



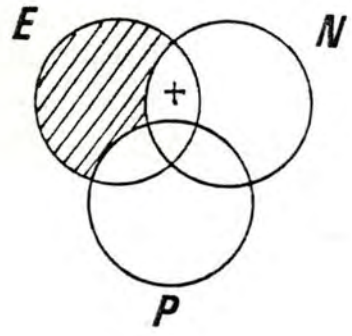
5.1.6. Limites de la représentation graphique.

La représentation par diagrammes permet de tester, dans certaines limites, la validité d'inférences dont les prémisses n'ont pas une des quatre structures prédéfinies.

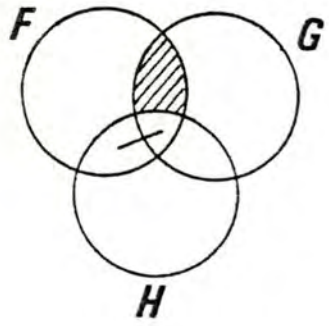
Par exemples :

toute personne à l'est de la piste est ou négligée, ou pauvre;
quelques personnes à l'est de la piste ne sont pas pauvres;
donc, quelques personnes négligées ne sont pas pauvres.

Graphiquement :



Tous les témoins (F) qui détiennent des actions dans l'entreprise (G) sont des employés (H);
quelques témoins (F) sont des employés (H) ou détiennent des actions dans l'entreprise (G);
donc, quelques uns des témoins (F) sont des employés (H).



Mais ces inférences ne contiennent toujours que 3 termes.

Voici une inférence nettement plus compliquée, empruntée à Lewis Caroll :

> aucun chaton qui aime le poisson n'est réfractaire à l'étude;
> aucun chaton sans queue n'est prêt à jouer avec un gorille;
> les chatons moustachus aiment toujours le poisson;
> aucun chaton amoureux de l'étude n'a les yeux verts;
> aucun chaton n'a de queue s'il n'est moustachu;
> donc, aucun chaton aux yeux verts n'est prêt à jouer avec un gorille.

Nous identifions six termes différents; le test de validité exige six diagrammes et la représentation devient beaucoup trop ardue.

5.2. Les axiomes A A A et A I I de la première figure.

5.2.1. Réduction des prémisses.

Au moyen des deux seuls syllogismes A A A et A I I de la 1ère figure, pris comme axiomes, nous pouvons retrouver tous les syllogismes valides.
Il est en effet possible de réduire chacun des syllogismes valides à une des formes :

A A A (1ère fig.)

Tout T est G
Tout P est T

Tout P est G

A I I (1ère fig.)

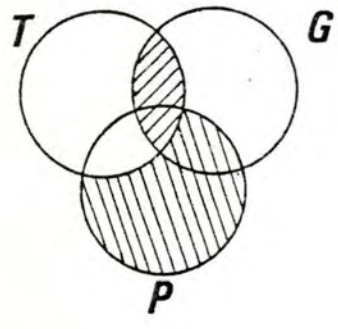
Tout T est G
Quelque P est T

Quelque P est G

1ère figure :
==========

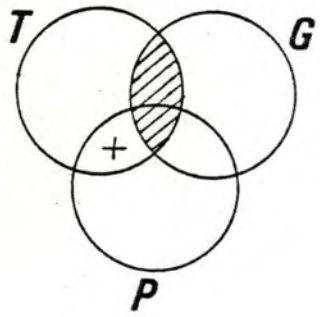
E A E (—> A A A 1ère fig.)

aucun T n'est G ——————> tout T est non G
tout P est T ——————> tout P est T
---
aucun P n'est G <—————— tout P est non G



E I O (—> A I I 1ère fig.)

aucun T n'est G	——————>	tout T est non G
quelque P est T	——————>	quelque P est T
quelque P n'est pas G	<——————	quelque P est non G



2e figure :
==========

E A E (—> A A A 1ère fig.)

aucun G n'est T	——————>	tout T est non G
tout P est T	——————>	tout P est T
aucun P n'est G	<——————	tout P est non G

A E E (—> A A A 1ère fig.)

tout G est T	——————>	tout non T est non G
aucun P n'est T	——————>	tout P est non T
aucun P n'est G	<——————	tout P est non G

E I O (—> A I I 1ère fig.)

aucun G n'est T ——————> tout T est non G
quelque P est T ——————> quelque P est T
_____ _____
quelque P n'est pas G <—————— quelque P est non G

A O O (—> A I I 1ère fig.)

tout G est T ——————> tout non T est non G
quelque P n'est pas T ——————> quelque P est non T
_____ _____
quelque P n'est pas G <—————— quelque P est non G

3e figure :
==========

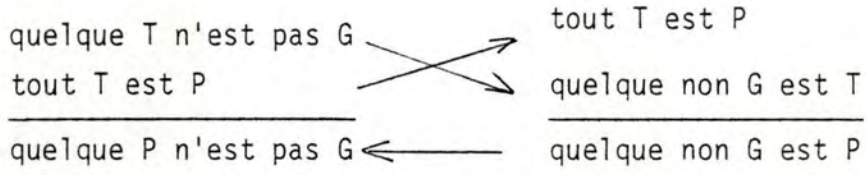
I A I (—> A I I 1ère fig.)

quelque T est G tout T est P
tout T est P quelque G est T
_____ _____
quelque P est G <—————— quelque G est P

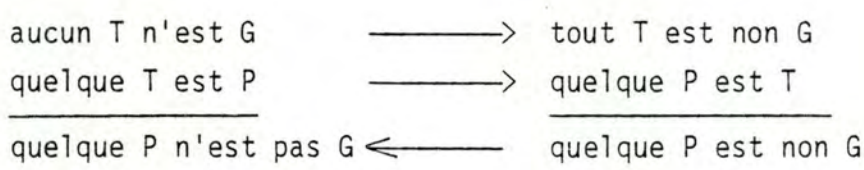
A I I (—> A I I 1ère fig.)

tout T est G ——————> tout T est G
quelque T est P ——————> quelque P est T
_____ _____
quelque P est G <—————— quelque P est G

O A O (—> A I I 1ère fig.)

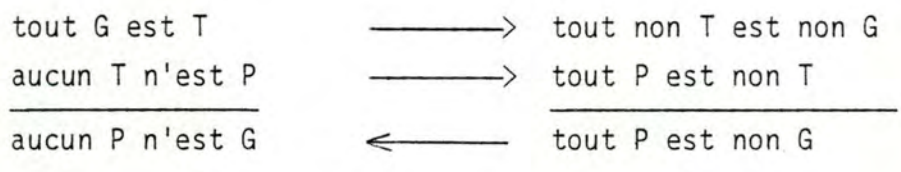


E I O (—> A I I 1ère fig.)

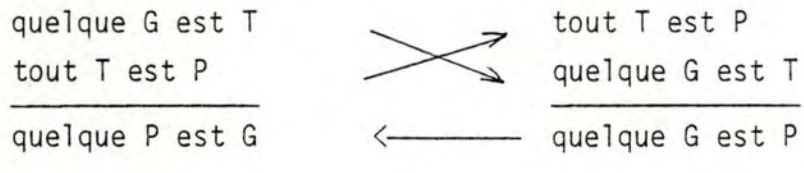


4e figure :
==========

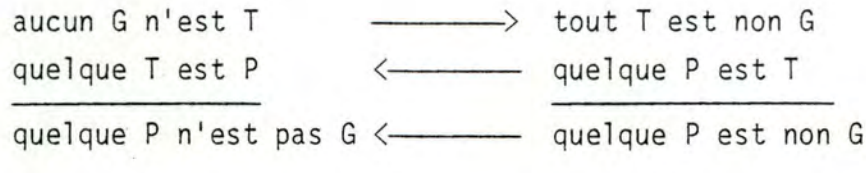
A E E (—> A A A 1ère fig.)



I A I (—> A I I 1ère fig.)



E I O (—> A I I 1ère fig.)



5.2.2. Recherche de la conclusion des syllogismes.

Nous pouvons vérifier la validité de tout syllogisme de notre classification, soit en utilisant le tableau du paragraphe 1.4, soit en utilisant les diagrammes, ou encore en réduisant le syllogisme à un des syllogismes AAA ou AII 1e figure).

Nous aimerions qu'un algorithme nous donne la conclusion à deux prémisses A, E, I, ou O ayant un terme commun. Pour ce faire, nous pouvons utiliser le tableau du paragraphe 1.4. sous forme d'un tableau de décision.
Nous pouvons également nous servir des deux axiomes en exécutant ces cinq étapes :

1. Observer la forme des deux prémisses.
   Si une des formes est un I ou un O, transformer les prémisses en A I ou I A.
   Sinon, transformer les prémisses en A A.
   Ces transformations se font en utilisant les propriétés (a) du paragraphe 1.2.

2. Eventuellement, inverser les deux prémisses de façon à avoir une forme A comme première prémisse.

3. Utiliser les propriétés (b) du paragraphe 1.2. pour obtenir le moyen terme à la bonne place :

$$\left\{\begin{array}{lll} T & - & x \\ y & - & T \end{array}\right\}$$

4. Tirer la conclusion en utilisant A A I ou A I I (1ère fig.).

5. Utiliser les propriétés (b) pour obtenir le petit terme P en première place.

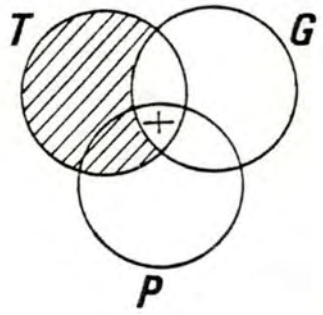
5.3. Limite de cette analyse.

Comme nous l'avons déjà fait remarquer, un test de validité devient vite très difficile si les prémisses contiennent plus de trois termes.
En outre, la classification des syllogismes est forcément limitée :

1. Nous avons imposé une conclusion P - G alors qu'il peut exister des conclusions T - G ou T - P également vraies et non redondantes avec l'une ou l'autre prémisse .

Tout T est G
Quelque P est T

Quelque P est G



Nous voyons que la conclusion 'Quelque T est G' qui ne répète aucune des deux premières prémisses, fournit aussi une inférence valide.

2. D'autres syllogismes sont déductibles des syllogismes de la classification mais n'ont pas été retenus du fait de leur conclusion G - P.

Tout T est G
Aucun P n'est T
Il y a des T

Quelque G n'est pas P

Cette forme est équivalente à la forme E A,
'il y a des T', O
si l'on échange G et P et si l'on inverse l'ordre
des deux premières prémisses.

3. La classification, ainsi que le diagramme de Venn, deviennent totalement inefficaces à représenter des prémisses lorsqu'elles se compliquent par des fonctions de vérité, comme :

si tous les F qui sont G sont H, alors quelques F
ne sont pas G
tous les F sont G ou tous les F sont H

Si nous voulons tester la validité d'inférences plus complexes que celles décrites dans ce chapitre, nous devons approfondir notre étude.
L'introduction de la quantification, dès le chapitre suivant, remédiera aux problèmes évoqués ici.

* * *

CHAPITRE III.

LOGIQUE DES PREDICATS

La logique des propositions considérait que chaque proposition devait être prise dans sa totalité, mais qu'elle pouvait se réduire à 'tout ou rien' losqu'on lui attribuait la valeur 1/0 selon qu'elle était vraie ou fausse. Un ensemble de propositions était susceptible d'être assimilé à des propositions individuelles articulées autour d'opérateurs logiques bien spécifiques qui déterminaient ainsi le type des liaisons qui les unissaient.

Les propositions étaient ainsi associées sous forme de prémisses (et de conclusions) pour constituer un énoncé dont la valeur de vérité dépendait des valeurs binaires 1/0 attribuées aux variables proportionnelles, et des opérations logiques effectuées sur ces valeurs.
On comprenait, ainsi, assez rapidement à la lecture des énoncés, qu'il n'était pas toujours possible de les exprimer sous des formes orthodoxes, même par des contorsions laborieuses. Ces acrobaties grammaticales d'ailleurs ne se font pas sans provoquer des distorsions plus ou moins importantes du jugement sous-tendu aux propositions.

L'introduction de la quantification, sans être parfaite, va permettre d'améliorer la décomposition des formes générales proposées en expressions plus fines et plus nuancées.

I. SCHEMA DE TERMES.

Une phrase élémentaire ne comporte qu'un sujet et qu'un prédicat, comme dans l'exemple "Joseph est agriculteur".

Dans le présent paragraphe, nous nous efforçons de distinguer les sujets des prédicats. Cette distinction est le premier pas vers une analyse plus fine des propositions; elle nous amènera à représenter prédicats et sujets par des variables, ainsi qu'à réinterpréter dans ce nouveau contexte, la notion de validité.

Si nous assignons au sujet "Joseph" le symbole x_1 et si nous représentons le prédicat "est agriculteur" par le symbole F, la phrase précitée peut se représenter par Fx_1.
Imaginons le domaine des sujets possibles; nous pouvons considérer qu'il forme un ensemble E d'individus ou d'objets x_i :

$$E = [\, x_1, x_2, \ldots x_n \,]$$

Supposons que cet ensemble comprenne les constantes d'individus :

x_1 = Joseph

x_2 = Maurice

x_3 = Léon

nous pouvons ouvrir le domaine des sujets à des ensembles autres que celui des gens (x_1, x_2, x_3) de la terre, comme par exemple à celui des gens de l'armée (y_1, y_2, ... y_n) et des hommes de lettres (z_1, z_2, ... z_n).
L'ensemble des constantes d'individus (x_1, x_2, x_3, y_1, y_2, ... y_n, z_1, z_2, ... z_n), forme l'univers de discours.

Nous pouvons, à présent, traduire les propositions composées à l'aide des éléments de l'univers de discours.

Par exemple : "Raymond n'est pas éleveur et (Joseph est agriculteur ou Léon est apiculteur)"
s'écrira, en assignant le symbole y_1 à "Raymond" :

$$G'y_1 \wedge (Fx_1 \vee H x_3)$$

Ce schéma est effectivement équivalent à la proposition ci-dessus, si nous substituons aux termes F, G, H (appelés également variables de prédicat), les prédicats :

"est agriculteur" = F
" est éleveur" = G
" est apiculteur" = H

Notons qu'il est possible de traduire "Joseph n'est pas un homme de lettres" par $M'x_1$, en substituant le prédicat "est un homme de lettres", au terme M.

Si le sujet est le même pour tous les prédicats intervenant dans le schéma : $G'x \wedge (Fx \vee H x)$, nous avons avantage à le réécrire plus simplement :

$$\left[G' \wedge (F \vee H) \right] x$$

Le schéma entre crochets est un schéma de terme : il s'agit d'un schéma constitué de termes liés par des fonctions de vérité.
Un schéma de terme peut être vrai ou faux de différents objets de l'univers : par exemple, avec les mêmes substitutions que précédemment, le schéma FG' sera vrai de tous les agriculteurs non éleveurs.

Examinons dans l'univers (x_1, x_2, x_3) des gens de la terre, la proposition composée de deux propositions simples p, p' :

$$p \vee p' \qquad (1)$$

Nous savons que ce schéma est vérifonctionnellement valide.

Quelle que soit la proposition substituée à p, elle se compose d'un quelconque des sujets x_1, x_2 et x_3, ainsi que d'un prédicat que nous représente-

-rons par la variable F.
Nous pouvons donc aussi bien écrire :

(F V F') x (2)

où x est une variable représentant n'importe lequel des individus de l'univers de discours et où n'importe quelle interprétation peut être donnée à la variable de prédicat F.

L'interprétation d'une lettre de terme F se fait en fixant dans l'univers de discours les éléments dont F doit être tenu pour vrai.
Dans ce cas-ci, F peut être interprété comme vrai de x_1 et faux de x_2, x_3 (ce serait, par exemple, le cas en lui substituant le prédicat "est agriculteur").
Il peut également être interprété comme vrai de tous les éléments de l'univers (par exemple en lui substituant le prédicat "est un paysan").
Nous pouvons encore l'interpréter :

faux de tous les éléments de l'univers;

vrai de x_2 et faux de x_1, x_3 ;

vrai de x_3 et faux de x_1, x_2 ;

vrai de x_1, x_2 et faux de x_3 ;

vrai de x_1, x_3 et faux de x_2 ;

ou vrai de x_2, x_3 et faux de x_1.

Puisque (2) est une expression équivalente à (1), nous sommes tout naturellement amenés à dire que le schéma de terme (F V F') est valide si et seulement si il est vrai de tous les objets de n'importe quel univers, pour toutes les interprétations de ses lettres de termes.
En fait, cette définition se généralise à tous les schémas de termes.

Remarquons que le schéma (F V F') a la même structure vérifonctionnelle que le schéma booléen (p V p' dans le cas présent) dont il dérive et ceci est vrai pour tous les schémas de termes (c'est pourquoi on les appelle parfois des schémas de termes booléens).
Par conséquent, la validité d'un schéma de termes peut se tester de la même façon que celle d'un schéma vérifonctionnel : à l'aide d'une table de vérité.

F	*F'*	*F V F'*
1	0	1
0	1	1

Un schéma de terme sera dit consistant s'il est vrai de quelque objet de quelque univers, pour quelque interprétation de ses lettres.
(Par exemple, le schéma FF' est inconsistant).
Un schéma de terme en implique un autre si toute interprétation rend le second schéma vrai de tout ce dont le premier est vrai.

Propriétés des schémas de termes.

(1) Si S_1, ..., S_n sont des schémas de terme booléen et si chacun d'eux pris individuellement est consistant, et si x_1, ... x_n sont des objets différents quelconques, alors dans l'univers constitué de x_1, ..., x_n, il existe une interprétation des termes (F, G, H, ...) composant les schémas qui en même temps rendent S_1 vrai de x_1, S_2 vrai de x_2, etc ...

En effet :

Puisque S_i est consistant, il existe une substitution de 'vrai' et de 'faux' aux termes F, G, ... qui réduit S_i à 'vrai' (pour tout i de 1 à n).
Interprétons F comme vrai de x_i chaque foi que 'vrai a été substitué à F pour réduire S_i à 'vrai'.
En faisant de même avec G, H, ..., on construit une interprétation qui rend le schéma S_i vrai de x_i (pour tout i de 1 à n).

Par exemple : $S_1 = FG'$; $S_2 = GH'$; $S_3 = F'H$

S_1 est vrai en substituant 'vrai' à F et 'faux' à G.

On interprète donc F et H (mais non G) comme vrai de x_1.

Suivant le même procédé, on interprète F et G (mais non H) comme vrai de x_2; on interprète H et G mais non F comme vrai de x_3.

Quand n = 1, nous avons les propriétés particulières :

(2) Si un schéma de terme booléen est consistant, alors pour un univers constitué d'un objet unique quelconque, il existe une interprétation de F, G, ... qui rend le schéma vrai de cet objet,

d'où le corollaire :

(3) si un schéma de terme booléen n'est pas valide, alors dans un univers constitué d'un objet unique quelconque, il existe une interprétation de F, G, ... qui rend le schéma faux de cet objet

et l'on a encore :

(4) si un schéma de terme booléen n'en implique pas un autre, alors dans un univers constitué d'un objet unique quelconque, il existe une interprétation de F, G, ... qui rend le premier schéma vrai de cet objet et le second faux du même objet.

II. LES QUANTIFICATEURS.

2.1. Le quantificateur existentiel ($\exists$ x)

Considérons que l'univers de discours est l'ensemble des hommes de lettres et interprétons F comme "est un romancier";

G comme "est un poète";

H comme "est un cinéaste".

Cet univers contient les individus :

[Baudelaire, Vian, Sand, Hitchkock, ...]

Nous savons déjà comment schématiser l'énoncé "Baudelaire est un poète" : il suffit d'associer le symbole x_1 au sujet "Baudelaire" et d'écrire : $G\ x_1$.
Si nous voulons dire qu'il existe un poète, nous écrirons $(\exists\ x)\ G\ x$ ou plus simplement $\exists\ G$.
$(\exists\ x)\ F\ G\ x$ signifiera "Il existe des romanciers poètes" (un ou plusieurs).
De la même façon, "Il n'y a pas de cinéaste poète" s'écrira :
$-\ (\exists\ x)\ G\ H\ x$ ou encore $-\ \exists\ G\ H$.

Il est très important de faire, dès à présent, la distinction entre le symbole x_1 du schéma $G\ x_1$ et le symbole x du schéma $(\exists\ x)\ G\ x$.
Dans le second schéma, le symbole est lié par un quantificateur : x est alors une variable muette; il ne désigne pas Baudelaire, ni aucune personne en particulier : $(\exists\ x)\ G\ x$ signifie que Baudelaire, Vian, Sand ou Hitchkock, ou ... (1 ou plusieurs de ces personnes) est un poète.
Un schéma formé de ' $\exists$ ' et suivi d'un schéma de terme (booléen) s'appelle un schéma d'existence (booléen).
Un schéma d'existence est valide s'il se révèle vrai dans tous les univers non vides, pour toutes les interprétations des lettres de termes.

> Il est nécessaire d'écarter l'univers vide de cette définition : $\exists\ F$ et $\exists\ F'$ y sont toujours faux et le schéma $\exists\ F\ V\ \exists\ F'$ n'y est pas valide, contrairement à tout autre univers non vide.

Par définition, un schéma d'existence est consistant s'il est vrai pour une interprétation des lettres de termes dans un univers non vide.

Propriétés des schémas d'existence booléens.

(5) Un schéma d'existence booléen est valide si et seulement si son schéma de terme est valide.

En effet :

Si le schéma de terme est valide, alors toute interprétation le rend vrai de tout objet de l'univers, et donc le schéma d'existence est valide.
Si le schéma de terme n'est pas valide, alors d'après (3), il existe une interprétation qui rend le schéma faux du seul objet constituant un certain univers et pour cette interprétation, le schéma d'existence est faux.

(6) Un schéma d'existence booléen est consistant si et seulement si son schéma de terme est consistant.

En effet :

Si le schéma de terme est consistant, alors d'après (2), il existe une interprétation qui rend le schéma vrai de quelque chose et par conséquent rend le schéma d'existence vrai.
Si le schéma de terme est inconsistant, toute interprétation le rend faux de tous les objets et par là même, rend faux le schéma d'existence.

(7) Un schéma d'existence booléen en implique un autre si et seulement si le premier schéma de terme implique le second.

En effet :

Si le premier schéma de terme implique le second, alors toute interprétation qui rende le premier schéma de terme vrai de quelque chose rendra l'autre vrai de la même chose. Toute interprétation qui rend le premier schéma d'existence vrai rendra donc le second vrai et ainsi, le premier implique le second.

Si le premier schéma de terme n'implique pas le second, alors d'après (4), il existe une interprétation qui rend le premier schéma de terme vrai du seul objet d'un certain univers et rend le second faux du même objet. Cette interprétation rend le premier schéma d'existence vrai et le second faux et ainsi le premier n'implique pas le second.

(8) Une conjonction de schémas d'existence booléens est consistante aussi longtemps que chacun d'eux pris séparément est consistant.

En effet :

Soient E_1, ... E_n, des schémas d'existence booléens quelconques, séparément consistants, et soient respectivement S_1, ... S_n, les schémas de terme booléens figurant dans chacun des schémas d'existence.
D'après (6), chaque S_i est consistant.
D'après (1), il existe alors une interprétation qui rend chacun d'eux vrai d'un objet différent. Cette interprétation rend vrais tous les schémas d'existence E_1, ... E_n.

(9) Si un schéma d'existence booléen est impliqué par une conjonction de schémas d'existence booléens, c'est qu'il est impliqué par l'un d'eux isolément.

En effet :

Soient E et E_1, ... E_n, des schémas d'existence booléens et supposons qu'aucun des schémas E_1, ... E_n n'implique E; il faut montrer que leur conjonction n'implique pas E.
Soient S et S_1, ... S_n, les schémas de termes figurant dans E, E_1, ... E_n.
D'après (7), puisque E_1 n'implique pas E, S_i n'implique pas S.
En conséquence, la conjonction de S_i avec la négation de S est consistante pour tout i.
D'après (1), il existe alors une interprétation qui rend chacun de ces n conjonctions vraie d'un objet différent.
Comme S est nié dans chaque conjonction, l'interprétation rend S faux des n objets, alors qu'elle rend S_1, ... S_n vrai chacun d'au moins un objet.
Dans cette interprétation, E_1, ... E_n deviennent tous vrais et E faux : la conjonction de E_1, ... E_n n'implique pas E.

(10) '∃ F V G' est équivalent à '∃ F V ∃ G'

('(∃ x)(F x V G x)' est équivalent à '(∃ x)F x V (∃ x) G x')

Il est évident que certaines personnes sont des cinéastes ou des romanciers si et seulement si certaines personnes sont des cinéastes ou certaines personnes sont des romanciers. (Il n'est pas exclu qu'il y ait à la fois des cinéastes et des romanciers ou qu'il y ait des cinéastes romanciers).

Remarque : ∃ F G n'est pas équivalent à ∃ F ∧ ∃ G.
Par exemple, il existe des choses rondes et des choses carrées ne signifie pas qu'"il existe des carrés ronds".

2.2. Le quantificateur universel (x).

Le quantificateur universel (x) correspond aux mots "chaque chose x de l'univers de discours est telle que".
Si, dans l'univers des hommes de lettres, nous voulons dire que tous sont des romanciers, nous écrirons :

(x) F x

Comme pour un quantificateur existentiel, le quantificateur universel porte sur une variable muette; le schéma ci-dessus signifie :

Baudelaire et Vian et Sand et Hitchkock et ... sont des romanciers.

Lien avec le quantificateur existentiel.

Dans l'univers des hommes de lettres, dire qu'il n'y a pas de romancier, (- ∃ F) revient à dire que tout homme de lettre est non romancier :

(x) F'x

D'autre part, nier que tous les hommes de lettres sont des romanciers (- (x) F x) revient à affirmer l'existence d'hommes de lettres non romanciers.

(du moins dans un univers non vide) :

$$\exists\, F'x$$

Donc, l'assemblage " $-\exists\, x$ " a le même effet que l'assemblage " $(x)-$ " et l'assemblage "$(\exists\, x)-$"a le même effet que l'assemblage " $-(x)$ "

Propriété.

Tout comme le quantificateur existentiel se distribue sur la disjonction, le quantificateur universel se distribue sur la conjonction :

$(x)(F\, x \wedge G\, x)$ est équivalent à $(x)\, F\, x \wedge (x)\, G\, x$

3. LES SCHEMAS QUANTIFICATIONNELS MONADIQUES.

Nous disposons dorénavant de trois types de représentation des propositions :

- les schémas vérifonctionnels qui contiennent des variables propositionnelles;
- les schémas de termes qui contiennent des termes et des variables d'individus mais dont les structures ne sont pas différentes des précédentes;
- et enfin, les schémas quantificationnels qui contiennent des termes et des variables, liées ou non par des quantificateurs.

Cette dernière catégorie de schémas est de loin la plus puissante; elle permet de transcrire des énoncés tels que "tous les chimpanzés sont omnivores" de la façon suivante :

Pour toute chose x de l'univers, si x est un chimpanzé
alors x est omnivore

En associant le prédicat "est un chimpanzé" au terme C et le prédicat "est omnivore" au terme O, nous obtenons :

$$(x)\ (C \supset O)\ x$$
$$(x)\ (C' + O)\ x$$

Cette écriture n'aurait pas été possible sans la distinction entre sujet et prédicat. Toutefois, cette distinction ne nous autorise pas à différencier un sujet d'un objet au sein d'un même énoncé :

par exemple :
tous les chimpanzés mangent des bananes
devient
$(x)\ (C \supset B)\ x$
en associant le prédicat "mange des bananes" au terme B

Mais il se peut qu'il soit nécessaire de séparer dans le prédicat le verbe "manger" du complément d'objet direct, pour traduire des phrases telles que :

"Il y a des fruits que tous les chimpanzés mangent"

Ceci ne peut se faire qu'en remplaçant "x mange y" par un terme portant sur deux variables : M x y.

On obtient alors :

$$(x)\,[C\,x \supset (\exists\, y)\, M\,x\,y].$$

Les termes portant sur plus d'une variable sont polyadiques (diadiques dans le cas de deux variables). Les autres sont monadiques.
Dans l'analyse qui suit, nous désirons nous limiter aux termes monadiques : les schémas polyadiques nécessitent en effet une analyse particulière dans la mesure où, à l'opposé des schémas monadiques, il n'est pas toujours possible de tester leur validité.

3.1. Schémas clos, schémas ouverts.

Puisque nous voulons envisager tous les schémas quantificationnels monadiques nous devons tenir compte de phrases telles que :

"Paul serait surpris si tous les hommes de lettres étaient cinéastes".

Celle-ci peut s'écrire :

$$(x)\,(H\,x) \supset p$$

où nous avons remplacé la proposition "Paul serait surpris" par p et où l'univers de discours est celui des hommes de lettres.
Si Paul est un élément de l'univers de discours, nous pouvons retranscrire p en Ly où les symboles y et L représentent respectivement le sujet "Paul" et le prédicat "serait surpris".

Nous obtenons :

$$(x)\ (Hx) \supset Ly \qquad \text{Libre} \qquad (1)$$

et nous pouvons schématiser l'énoncé

" Il existe quelqu'un (un homme de lettres, puisqu'il appartient à l'univers de discours) qui serait surpris si tous les hommes de lettres étaient des cinéastes"

de la façon suivante :

$$(\exists y)\ (x)\ (Hx \supset Ly) \qquad \text{Close} \qquad (2)$$

Dans le schéma (1), un des composants (Ly) contient une variable non quantifiée; une telle variable est dite libre; toutes les variables non libres sont liées par des quantificateurs.
Les phrases représentées par des schémas tels que (2) qui ne contiennent que des variables liées sont appelées "phrases closes" et les schémas correspondants sont des schémas clos. A l'inverse, si le schéma contient une variable libre, il s'agit d'un schéma ouvert.
En fait, seules les phrases closes possèdent une valeur de vérité; les phrases ouvertes ne sont ni vraies, ni fausses, mais peuvent être dites vraies ou fausses de différents objets de l'univers. Ainsi, le schéma (1) peut être vrai de Paul, mais faux de Pierre.

Les lettres schématiques p représentent des propositions non quantifiées : aussi bien des énoncés de la logique des propositions que des phrases ouvertes : nous les appellerons des lettres de phrases.
L'étude des schémas quantificationnels doit être suffisamment large que pour englober celle des schémas vérifonctionnels (tels que : $p \supset q$). Ces schémas sont en effet des cas particuliers de schémas quantificationnels, sans quantificateurs. Représentant des énoncés, ils possèdent une valeur de vérité et comptent donc pour des schémas quantificationnels clos.

3.2. Règles de passage.

Reprenons le schéma

$$(x)\ (Hx) \supset Ly \qquad (1)$$

et notons qu'il n'est pas équivalent au schéma

$$(x)\ [Hx \supset Ly] \qquad (2)$$

Ce dernier nous dit que "Paul serait surpris qu'il existe des hommes de lettres qui soient également des cinéastes".
La différence de signification des deux schémas, due à la portée différente du quantificateur universel, se traduit dans le langage courant par le pronom indéfini "chacun" pour une portée restreinte et par le pronom "quiconque" pour une portée large :

(1) signifie "Paul serait surpris si chaque homme de lettres était un cinéaste"

(2) signifie "Paul serait surpris si quiconque (si un homme de lettres, quel qu'il soit) était cinéaste".

Le second schéma contient dans la portée du quantificateur une variable libre y : nous disons que ce schéma est impur. Au contraire, dans le premier schéma, aucun quantificateur ne porte sur une expression contenant une variable libre : un tel schéma est pur.
Nous aimerions trouver la forme pure équivalente à (2) et plus généralement, trouver les formes pures équivalentes à une disjonction, une conjonction ou un c;nditionnel qui contiendrait une impureté dans une de ses composantes.
Ces schémas purs s'obtiennent par 8 règles de passages.
L'énoncé $(\exists x)\ (p \wedge Fx)$ est vrai pour les interprétations de la lettre de phrase p et du terme F qui rendent $(p \wedge Fx)$ vrai d'au moins une chose x.
Ce sont les mêmes interprétations qui rendent p vrai et F vrai de quelque

chose, si bien qu'on a l'équivalence :

$$(1)\quad (\exists x)\,(p \wedge Fx) \longleftrightarrow p \wedge (\exists x)\,Fx$$

de la même manière :

$$(2)\quad (x)\,(p \wedge Fx) \longleftrightarrow p \wedge (x)\,Fx$$

Nous pouvons transformer (1) : $-(x)\,(p' \vee F'x) \longleftrightarrow p \wedge (-x)F'x$

$$(x)\,(p' \vee F'x) \longleftrightarrow p' \vee (x)\,F'x$$

$$(3)\quad (x)\,(p \vee fx) \longleftrightarrow p \vee (x)\,Fx$$

et transformer (2) : $-\exists\,(p' \vee F'x) \longleftrightarrow p \wedge -\exists\,F'x$

$$\exists\,(p' \vee F'x) \longleftrightarrow p' \vee \exists\,F'x$$

$$(4)\quad (\exists x)\,(p \vee Fx) \longleftrightarrow p \vee (\exists\, x)Fx$$

De (3) et (4), nous déduisons :

$$(5)\quad (\exists x)\,(p \supset Fx) \longleftrightarrow p \supset (\exists x)\,Fx$$

$$(6)\quad (x)\,(p \supset Fx) \longleftrightarrow p \supset (x)\,Fx$$

Nous pouvons encore utiliser (4) : $(\exists x)\,(Fx \supset p)$

$$(\exists x)\,(F'x \vee p)$$

$$(\exists x)\,(F'x) \vee p$$

$$-(x)\,(Fx) \vee p$$

et obtenir :

$$(7)\quad (\exists x)\,(Fx \supset p) \longleftrightarrow (x)\,Fx \supset p$$

En utilisant (3), on peut démontrer :

(8) $(x)\,(Fx \supset p) \longleftrightarrow (\exists x)\,Fx \supset p$

remarque : il n'existe aucune règle de passage d'un quantificateur à travers le symbole '≡' du biconditionnel.
Aucune de ces expressions

$(x)\,(p \equiv Fx)$; $p \equiv (x)\,Fx$; $(\exists x)\,(p \equiv Fx)$; $p \equiv (\exists x)\,Fx$

n'est équivalente.

Le lecteur s'en convaincra en substituant 'faux' puis 'vrai' à la lettre de phrase p.

Par exemple en supposant que p est 'faux'

$(x)\,(p \equiv Fx)$ devient $(x)\,(\text{faux} \equiv Fx)$

$(x)\,(F'x)$

$-\exists\, F$

le schéma résultant à la même valeur de vérité que le schéma $p \equiv (\exists x)\,Fx$

$\text{faux} \equiv (\exists x)\,Fx$

$-\exists\, F$

Par contre, si p est 'vrai',

$(x)\,(p \equiv Fx)$ devient $(x)\,Fx$

Ce schéma n'a pas la valeur de vérité de

$p \equiv (\exists x)\,Fx$

mais celle de $p \equiv (x)\,Fx$

$\text{vrai} \equiv (x)\,Fx$

$(x)\,Fx$

3.3. Formes pures.

Nous avons déjà rencontré la notion de schéma pur; nous avons vu comment les huit règles de passage purifient des schémas réduits à une simple conjonction, disjonction ou conditionnel comprenant une impureté dans une de leurs deux composantes.
Nous expliquons à présent comment ces règles, judicieusement appliquées, peuvent purifier tout schéma quantificationnel monadique.
Rappelons la définition :

> Une forme pure est une expression où chaque quantificateur porte sur une fonction de vérité dont chaque composant présente des occurrences libres de la variable du quantificateur.

Théorème : Il est toujours possible de purifier un schéma quantificationnel monadique.

Supposons que le schéma contienne le sous-schéma existentiel :

$$(\exists x)\left[(Fx \vee p) \wedge (Gx \vee q)\right]$$

mettons sous forme normale disjonctive le schéma qui suit le quantificateur

$$(\exists x)\left[\, Fx\, Gx \vee q\, Fx \vee p\, Gx \vee pq \,\right]$$

Les impuretés sont p et q où x fait défaut.
Par distributivité et application de la règle (1) :

$$(\exists x)\,(Fx\, Gx) \vee q\ (\exists x)\, Fx \vee p\ (\exists x)\, Gx \vee pq$$

Si le schéma contient le sous-schéma universel :

$$(x)\left[(Fx \vee p) \wedge (Gx \vee q)\right]$$

Laissant le schéma sous forme normale conjonctive, nous appliquons la propriété de distributivité et la règle (3) :

$$((x)\ Fx\ V\ p) \wedge ((x)\ Gx\ V\ q)$$

Nous savons ainsi purifier un sous-schéma contenant un seul quantificateur existentiel ou universel.

Ces deux procédures permettent-elles de purifier tout le schéma ?

Elles y suffisent si chaque sous schéma possède un seul quantificateur et est relié aux autres par une fonction de vérité.

Elles sont incomplètes dans un cas général.

En effet, si le schéma est clos, et si une des impuretés rencontrées dans un sous schéma contient une variable (par exemple $\left.\begin{matrix} p = Fz \\ q = Hy \end{matrix}\right)$, celle-ci doit être liée.

- soit que le quantificateur qui la lie ne porte sur aucune impureté (par exemple, p = (z) Fz; q = $(\exists y)$ Hy): ce cas ne pose pas de problème;
- soit que la formule est régie par des quantificateurs emboîtés, par exemple : $(\exists y)$ (z) $(\exists x)$ (Fx V Fz) $\wedge$ (Gx V Hy).
  Mais alors les règles (1) - (8) permettent de refouler les quantificateurs vers l'intérieur et de cette façon font porter chaque quantificateur sur la variable qui le concerne :

  $(\exists y)$ (z) $(\exists x)$ [Fx Gx V Fx Hy V Fz Fx V Fz Hy]

  Par distributivité et par la règle (1) :

  $(\exists y)$ (z) [$(\exists x)$ (Fx Gx) V $(\exists x)$ (Fx) Hy V Fz $(\exists x)$ Fx V Fz Hy]

  Par les règles (2) et (3) :

  $(\exists y)$ [$(\exists x)$ (Fx Gx) V $(\exists x)$ (Fx) Hy V (z) Fz $(\exists x)$ Fx V (z) (Fz) Hy]

Et enfin, par les règles (1) et (4)

$(\exists x)\ (Fx\ Gx)\ V\ (\exists x)\ (Fx)\ (\exists y)\ (Fy)\ V\ (z)\ (Fz)\ (\exists x)\ Fx$

$V\ (z)\ (Fz)\ (\exists y)\ Hy)$

Puisque le schéma initial est clos, le schéma résultant de ces opérations ne contiendra que des variables liées à des quantificateurs ne portant sur aucun impureté.

La procédure générale de purification d'un schéma monodique est la suivante :

- partout où se rencontre une quantification impure, on met la formule régie par le quantificateur sous forme normale disjonctive ou conjonctive (suivant que la quantification est existentielle ou universelle);
- on applique les règles (1) à (8) qui sont applicables de façon à refouler les quantificateurs vers l'intérieur.

Par exemple, purifions :

$$(x)\ \left[(\exists y)\ (Fx \equiv Gy)\ V\ (\exists y)\ Fy\right] \wedge\ (x)\ (Fx\ V\ Gx)$$

par la règle (3) :

$$\left[(x)\ (\exists y)\ (Fx \equiv Gy)\ V\ (\exists y)\ Fy\right] \wedge\ (x)\ (Fx\ V\ Gx)$$

mettons la partie $(Fx \equiv Gy)$, qui suit $(\exists y)$ et comprend l'impureté Fx, sous forme normale disjonctive et distribuons $(\exists y)$

$$\left[\ (x)\ \left[(\exists y)(Fx \wedge Gy)\ V\ (\exists y)(F'x \wedge G'y)\right]\ V\ (\exists y)\ Fy\right]\ (x)(Fx\ V\ Gx)$$

En appliquant deux fois la règle (1), il vient :

$$\left[\ (x)\ \left[Fx\ (\exists y)\ Gy\ V\ F'x\ (\exists y)\ G'y\right]\ V\ (\exists y)\ Fy\ \right] \wedge (x)(Fx\ V\ Gx)$$

Mettons la partie qui suit le premier quantificateur universel sous forme conjonctive :

$$\Big[(x)\ \big[(Fx \vee (\exists y)\ G'y) \wedge ((\exists y)\ Gy \vee F'x) \wedge ((\exists y)\ Gy \vee (\exists y)\ G'y)\big] \vee (\exists y)\ Fy\Big] \wedge (x)(Fx \vee Gx)$$

Appliquons la règle (2) :

$$\Big[\big[(x)\ \big[(Fx \vee (\exists y)G'y) \wedge (\exists y)(Gy \vee F'x)\big] \wedge \big[(\exists y)\ Gy \vee (\exists y)\ G'y\big]\big] \vee (\exists y)\ Fy)\Big] \wedge (x)\ (Fx \vee Gx)$$

Distribuons le quantificateur (x) :

$$\Big[\big[(x)\ (Fx \vee (\exists y)\ G'y) \wedge (x)\ ((\exists y)\ Gy \vee F'x) \wedge ((\exists y)\ Gy \vee (\exists y)G'y)\big] \vee (\exists y)\ Fy)\Big] \wedge (x)\ (Fx \vee Gx)$$

En appliquant deux fois la règle (4) :

$$\Big[\big[((\exists y)\ G'y \vee (x)\ Fx) \wedge ((\exists y)\ Gy \vee (x)\ F'x) \wedge ((\exists y)\ Gy \vee (\exists y)\ G'y)\big] \vee (\exists y)\ Fy\Big] \wedge (x)\ (Fx \vee Gx)$$

Nous avons purifié ainsi le schéma initial.

3.4. Forme prénexe.

La notion de schéma prénexe est duale de celle de schéma pur : une forme prénexe est une expression où les quantificateurs se trouvent tous en tête.

Théorème : Il est toujours possible de mettre sous forme prénexe un schéma quantificationnel.

Quelques transformations préalables du schéma sont nécessaires.

1. Puisqu'il n'existe pas de règle de passage d'un quantificateur à travers le symbole du biconditionnel, nous devons éliminer toute occurrence de ' ≡ ' qui unirait des schémas quantificationnels.

2. Il nous faut modifier les lettres des variables liées, avant d'appliquer les règles de passage.

   En effet : $Fx \wedge (\exists x)\, Gx \not\leftrightarrow (\exists x)\, (Fx \wedge Gx)$

   Par contre, en modifiant le nom de la variable liée par $\exists$, nous avons :

   $$Fx \wedge (\exists y)\, Gy \longleftrightarrow (\exists y)\, (Fx \wedge Gy)$$

Ces préparatifs terminés, la forme prénexe s'obtient en refoulant les quantificateurs à l'extérieur du schéma, par application des 8 règles de passage.

Exemple : $(x)\, Fx \supset ((x)\, Gx \equiv (x)\, Hx)$

$(x)\, Fx \supset [((y)\, Gy \supset (z)\, Hz) \wedge ((w)\, Hw \supset (v)\, Gv)]$

Après une double application de la règle (7) :

$(x)\, Fx \supset [(\exists y)\, (Gy \supset (z)\, Hz) \wedge (\exists w)\, (Hw \supset (v)\, Gv)]$

En appliquant deux fois (1) :

$$(x)\ Fx \supset (\exists y)(\exists w)\ \big[(Gy \supset (z)\ Hz)(Hw \supset (v)\ Gv)\big]$$

En utilisant deux fois la règle (6), puis deux fois la règle (7) :

$$(x)\ Fx \supset (\exists y)(\exists w)(z)(v)\ \big[(Gy \supset Hz)(Hw \supset Gv)\big]$$

Finalement, grâce à (7), (5) et (6)

$$(\exists x)(\exists y)(\exists w)(z)(v)\ \big[Fx \supset (Gy \supset Hz)(Hw \supset Gv)\big]$$

remarque : Il se peut que les lois de distributivité fournissent plus rapidement une forme prénexe

Ainsi le schéma

$$(x)\ \big[(\exists y)(Fx \supset Gz) \vee (\exists z)\ Fz\big] \wedge (x)(F \vee H)\ x$$

est équivalent au schéma

$$(x)\ \big[\big[(\exists y)(Fx \supset Gy) \vee (\exists z)\ Fz\big] (Fx \vee Hx)\big]$$

Ensuite, par distributivité de '$\exists$ ' sur ' V ', nous obtenons :

$$(x)\ \big[(\exists y)\ \big[\big[(Fx \supset Gy) \vee Fy\big] \wedge (Fx \vee Hx)\big]\big]$$

Puis, par une seule application de la règle (1) :

$$(x)(\exists y)\ \big[((Fx \supset Gy) \vee Fy) \wedge (Fx \vee Gx)\big]$$

3.5. Test de validité sur une forme prénexe.

Nous sommes désormais en mesure de réécrire tout schéma quantificationnel dans une de ses formes pure, ou prénexe.
Il est temps de développer une procédure pour tester la validité d'un schéma monadique : nous le ferons pour chacune des formes, pure, et prénexe.

Théorème : Etant donné un schéma quantificationnel monadique, il est toujours possible de le mettre sous une forme prénexe où tous les quantificateurs universels sont en tête

C'est évident, puisque tout schéma monadique peut être purifié.
Une fois purifié, il ne contient plus de quantificateurs emboîtés et dès lors, nous pouvons ranger les quantificateurs dans l'ordre qu'il nous plaît, par les règles de passage (1) - (8) appliquées dans l'ordre adéquat.
Il est clair cependant, qu'il n'est pas obligatoire de passer au schéma purifié pour obtenir la forme recherchée.

Ainsi, le schéma

$$(x)\ \left[Gx(y)\ Fy \supset (\exists z)\ Fz\right]$$

se met directement sous la forme :

$$(x)(\exists z)(\exists y)\ \left[Gx\ Fy \supset Fz\right]$$

3.5.1. La procédure de test.

D'après la signification du quantificateur universel, il est évident que la question de la validité d'un schéma ouvert se ramène à celle de la validité d'un schéma clos obtenu en préfixant un quantificateur universel pour chaque variable libre.

Par exemple, la validité de $(\exists y)(\exists z)\ [Gx\ Fy \supset Fz]$ est la validité de
$(x)(\exists z)(\exists y)\ [Gx\ Fy \supset Fz]$ (1)
Par conséquent, tester la validité de ce schéma revient à tester la validité de $(\exists z)(\exists y)(Gx\ Fy \supset Fz)$ (2)
(Un tel schéma prénexe sans quantificateur universel est un schéma existentiel pur).
Si nous substituons le 'x' libre aux variables existentielles y, z, nous obtenons :

$$Gx\ Fx \supset Fx \qquad (3)$$

Le schéma (3) est obtenu par instantiation du schéma (2).
Nous voyons que (3) implique (2) qui lui-même implique (1).
Le schéma (3) étant valide, nous en déduisons que le schéma (2) et par là, le schéma (1) le sont également.

Cet exemple nous montre que si un schéma existentiel pur contient une variable libre, et si la substitution de cette variable aux variables existentielles rend le schéma vérifonctionnellement valide, alors le schéma existentiel pur est valide.
L'inverse est également vrai : le schéma pur est valide seulement si cette substitution rend le schéma vérifonctionnellement valide.
En effet, remplaçant les variables liées par la variable libre, nous finissons par obtenir un schéma portant sur une seule variable 'x', c'est-à-dire un schéma de terme.
Démontrons que si ce schéma n'est pas valide, le schéma existentiel d'où il provient ne peut pas l'être (pour faciliter la compréhension, nous raisonnons sur un exemple : $(x)(\exists y)(Fx \supset Fy\ Gx)$

Le schéma de terme $(Fx \supset Fx\ Gx)$ n'étant pas valide, il est faux d'un objet x_1 pour une certaine interprétation des termes.
Le schéma existentiel $(\exists y)(Fx_1 \supset Fy\ Gx_1)$ est une disjonction de schémas où la variable liée est remplacée par un élément de l'univers :

$$(Fx_1 \supset Fx_1\ Gx_1) \vee (Fx_1 \supset Fx_2\ Gx_1) \vee \ldots \vee (Fx_1 \supset Fx_n\ Gx_1)$$

c'est-à-dire, une disjonction de schémas vérifonctionnels :

$$(p \supset pq) \vee (p \supset rq) \vee \ldots \vee (p \supset sq)$$

Sachant qu'un de ces schémas $(p \supset pq)$ n'est pas valide et qu'il peut être obtenu à partir des autres en remplaçant une des variables propositionnelles par p (r, ..., ou s par p), nous en concluons qu'aucun des schémas de la disjonction n'est valide et donc la disjonction entière est non valide.

Plus généralement, si le schéma existentiel pur contient plus d'une variable libre :

> Un schéma existentiel pur est valide si et seulement si l'on obtient un schéma vérifonctionnellement valide en prenant la disjonction des résultats dus à la substitution des variables libres aux variables existentielles.

D'où la procédure suivante :

1. mettre le schéma sous une forme prénexe, avec les quantificateurs universels en tête; supprimer les quantificateurs universels;
2. construire autant de schémas que possible, en substituant les variables libres aux variables existentielles;
3. lier tous ces schémas par une disjonction;
4. tester la validité vérifonctionnelle du schéma obtenu en 3.

exemple : testons la validité de :

$$\left[(x)(M' + G' + F)x \wedge (\exists y)\, MF'y \wedge (\exists z)\, MHz\right] \supset (\exists v)\, MF'G'v$$

$$(y)(z)(\exists x)(\exists v)\ \left[\left[(M' + G' + F)\, x \wedge MF'y \wedge MHz\right] \supset MF'G'v\right]$$

En supprimant les quantificateurs universels, y et z deviennent des variables libres.

Il y a 4 façons de substituer y, z aux variables liées x et v :

y à x et y à v

z à x et z à v

y à x et z à v

z à x et y à v

La première de ces substitutions fournit le schéma :

$$[(M' + G' + F)y \wedge MF'y \wedge MHz] \supset MF'G'y$$

qui est valide puisqu'en notant $My = r$

$Gy = s$

$Fy = t$

$MHz = n$

on a :

$$(r' + s' + t).rt'.n \supset r\ s'\ t'$$

$$r' + t + s + n' + r\ s't'$$

$$r' + s + n' + t + t' = 1$$

1

3.6. Test de validité sur une forme pure.

Cette procédure de test se base sur un type particulier de schéma pur : les schémas purs ne contenant pas de variable libre. De tels schémas dérivent de schémas quantificationnels clos.
Nous verrons en effet que la procédure de test d'un schéma pur se réduit toujours au sous problème du test de validité d'un schéma quantificationnel pur et clos. Nous verrons aussi comment le test de validité d'un schéma quantificationnel clos se ramène toujours à celui d'un schéma vérifonctionnel booléen.

3.6.1. Les schémas d'énoncé booléen.

Reprenons le dernier exemple du paragraphe 3.3.
En écrivant '(x)' sous la forme '-(∃x)-' et en supprimant les variables liées (qui sont, rappelons-le, des variables muettes) :

$$\Big[\big[(\exists G' \vee -\exists F') \wedge (\exists G \vee -\exists F) \wedge (\exists G \vee \exists G')\big] \vee \exists F\Big] \supset -\exists F'G'$$

Les schémas de ce genre, qui sont des fonctions de vérité de schémas d'existences booléens s'appellent des schémas d'énoncé booléen (un tel schéma est valide s'il est vrai pour toutes les interprétations des lettres de termes dans tous les univers non vides).
Il est évident que tout schéma monadique, clos et ne contenant pas de lettre de phrase peut être transformé en un schéma d'énoncé booléen.

Propriétés :
==========

a) Un schéma d'existence booléen est valide ssi son schéma de termes est valide
(c'était le théorème (5)).

b) La négation d'un schéma d'existence booléen est valide ssi le schéma de terme est inconsistant.

En effet, la négation est valide ssi le schéma d'existence lui-même est inconsistant et donc, d'après (6), ssi le schéma de termes est inconsistant.

c) Une disjonction de négations de schémas d'existence booléens est valide ssi l'une des négations satisfait au test de validité précédent.
En effet, la disjonction des négations équivaut à la négation de la conjonction des schémas d'existence et celle-ci est valide uniquement au cas où la conjonction est inconsistante.
Mais d'après (7), cette conjonction est inconsistante uniquement au cas où l'un des schémas d'existence est inconsistant et par conséquent, uniquement au cas où l'une des négations de schéma d'existence est valide.

d) Un conditionnel existentiel est valide ssi le schéma de terme booléen figurant dans l'un des schémas d'existence de l'antécédent implique le schéma de terme figurant dans le conséquent. (Un conditionnel existentiel est un conditionnel dont l'antécédent est un schéma d'existence booléen ou une conjonction de tels schémas et dont le conséquent est un schéma d'existence).
En effet, le conditionel existentiel est valide ssi l'antécédent implique le conséquent. D'après (9), ce sera le cas ssi l'un des schémas d'existence de l'antécédent implique le conséquent.
D'après (7), cela sera le cas ssi le premier schéma de terme implique le second.

e) Une conjonction de schémas d'énoncé booléens ayant l'une quelconque des 4 formes précédentes est valide ssi chacun d'eux se révèle valide par les tests (a) - (d).

Théorème : Quel que soit un schéma d'énoncé booléen, il est toujours possible de décider si il est valide ou non.

démonstration :

Il est toujours possible de réduire un schéma d'énoncé S à une des 5 formes considérées en (a) - (e).
Pour ce faire, nous commençons par mettre S sous forme normale conjonctive (pour cela, nous traitons chaque négation de schéma d'existence figurant dans S comme un énoncé p, q, ...).
Ainsi, S prend la forme d'une conjonction de disjonction où les disjonctions sont des disjonctions de schémas d'existence ou des négations de schémas d'existence, ou les deux.
En utilisant la distributivité ($\exists f \vee \exists g = \exists (f \vee g)$, chacune des disjonctions comprend une ou plusieurs négations de schémas d'existence, mais au plus un schéma d'existence.
S'il s'agit d'une disjonction contenant une seule négation de schéma d'existence, elle tombe sous (b); si elle en contient plusieurs, elle tombe sous (c); si elle n'en contient aucune, mais qu'elle contient un schéma d'existence, elle tombe sous (a).
Enfin, s'il s'agit d'une disjonction de une ou plusieurs négations de schémas, et d'un seul schéma d'existence, elle équivaut à un conditionnel et tombe sous (d) (car $p'_1 \vee p'_2 \vee \ldots \vee p'_n \vee q$ équivaut à $p_1\, p_2 \ldots p_n \supset q$).

Pour une illustration, appliquons le test de validité à l'exemple :

$$(\neg \exists m\, g\, f') \wedge (\exists m\, f') \supset \exists m\, g'$$

après transformations :

$$\overline{(\neg \exists m\, g\, f' \wedge \exists m\, f')} \vee \exists m\, g'$$

$$\exists m\, g\, f' \vee \neg \exists m\, f' \vee \exists m\, g'$$

$$\exists (m\, g\, f' \vee m\, g') \vee \neg \exists m\, f'$$

Nous avons une seule négation de schéma d'existence :

$$\exists mf' \supset \exists(mgf' \vee mg')$$

Ce schéma d'énoncé booléen est valide ssi :

$$mf' \rightarrow mgf' + mg'$$

ou $m' + f + mgf' + mg' = 1$

$m' + g' + m = 1$,

ce qui est le cas; le schéma initial est donc valide.

Toujours à titre d'exemple, testons la validité de l'inférence suivante :

> Personne n'était présent (f) sauf les parents des acteurs (g). Si nul parent des acteurs (g) n'a aimé la pièce (h') alors nul parent d'acteurs (g) n'a de goût pour les fêtes scolaires (j). Donc, si d'aucun était présent (f) qui ont du goût pour les fêtes scolaires(j), alors quelque parent d'acteurs (g) a aimé la pièce (h).

En schématisant :

$$\left[(x)\ (f \supset g)\ x \ \wedge \left[(x)\ (g \supset h')x \supset (x)\ (g \supset j')x\right]\right] \supset \left[(\exists x)\ (fj)x \supset (\exists x)\ (gh)x\right]$$

Mettons ce schéma sous forme normale conjonctive :

$$\left[-\exists f\ g' \wedge (\exists gh \vee -\exists gj)\right] \supset (-\exists fj \vee \exists gh)$$

$$\exists fg' \vee (-\exists gh \wedge \exists gj) \vee -\exists fj \vee \exists gh$$

Sachant que $p + qr = (p + q)(p + r)$:

$$(\exists gh \vee -\exists fj \vee \exists fg' \vee -\exists gh)\ (\exists gh\ -\exists fj\ \exists fg'\ \exists gj)$$

Il reste à tester la validité des deux schémas de la conjonction.

$$\exists gh \vee -\exists fj \vee \exists fg' \vee -\exists gh = \exists(gh \vee fg') \vee -\exists fj \vee -\exists gh$$

$$\exists fj \wedge \exists gh \supset \exists(gh \vee fg')$$

d'après (d), ce schéma est valide ssi :

$$fj \rightarrow gh + fg'$$

ou $gh \rightarrow gh + fg'$ ce qui est minifestement le cas.

$\exists gh \vee - \exists fj \vee \exists fg' \vee \exists gj = - \exists fj \vee \exists (gh \vee fg' \vee gj)$

$\exists fj \supset \exists (gh \vee fg' \vee gj)$

d'après (d) ce schéma est valide ssi

$\overline{fj} + gh + fg' + gj = 1$

$f' + j' + g' + g = 1$ ce qui est manifestement le cas.

Donc d'après (e), l'inférence formulée par le schéma initial est valide.

Théorème $\quad \exists fg \rightarrow \exists f \wedge \exists g$

démonstration :

Testons la validité du conditionel : $\exists fg \supset \exists f \wedge \exists g$

$(- \exists fg \wedge \exists f) \wedge (- \exists fg \wedge \exists g)$

Ce schéma est valide car $fg \rightarrow g$ et $fg \rightarrow f$

Corollaire $\quad (x)\ f x \vee (x)\ g x \rightarrow (x)\ (f \vee g)\ x$

3.6.2. La procédure de test

1. Nous pouvons tester la validité d'1 schéma quantificationnel monadique clos lorsqu'il est dépourvu de lettres de phrases, en le réduisant à un schéma d'énoncé booléen et en utilisant les propriétés (a)-(e) décrites dans le paragraphe précédent 3.6.1.

2. Nous pouvons également tester la validité d'un schéma quantificationnel monadique ouvert et sans lettre de phrases.

   A titre d'exemple :

   $$\exists Fz \wedge (x)\ [(\exists y)\ Fy \supset Gx] \supset (\exists y)\ (Fy \wedge Gy)$$

   Par définition du quantificateur universel, il nous faut tester la validité du schéma clos :

   $$(z)\ [[Fz \wedge (x)\ [(\exists y)\ Fy \supset Gx]] \supset (\exists y)\ (Fy \wedge Gy)]$$

   Par application de la règle (8), de la règle (1) puis de la règle (6) :

$$\left[(\exists z)\, Fz \wedge \left[(\exists y)\, Fy \supset (x)\, Gx\right]\right] \supset (\exists y)\,(Fy \wedge Gy)$$

En transformant le quantificateur (x) et en éliminant les variables liées :

$$-\exists F \vee \left[\exists F \wedge \exists G'\right] \vee \exists FG$$

ou

$$(-\exists F \vee \exists FG \vee \exists F) \wedge (-\exists F \vee \exists FG \vee \exists G')$$

le premier schéma est valide puisque " $-\exists F \vee \exists F$ " est toujours vrai.

le second schéma est valide puisque :

$$F \rightarrow FG + G'$$

$$F' + F = 1$$

3. Examinons le problème de la validité d'un schéma quantificationnel monadique possèdant des lettres de phrases.
Ce schéma sera valide s'il est vrai pour chaque substitution de "vrai" et de "faux" à ses lettres de phrases.
Par exemple:

$$(x)\,(Fx \supset p) \equiv (\exists x)\, Fx \supset p$$

p = "vrai"

$$(x)\,(Fx \supset \text{"vrai"}) \equiv (\exists x)\, Fx \supset \text{"vrai"}$$

$$(x)(\text{"vrai"}) \equiv (\exists x) Fx \supset \text{"vrai"}$$

$$\text{"vrai"} \equiv \text{"vrai"}$$

"vrai" .

p = "faux"

$$(x)(Fx \supset \text{"faux"}) \equiv (\exists x) Fx \supset \text{"faux}$$

$$(x)\, F'x \equiv -\exists F$$

$$-\exists F \equiv -\exists F$$

"vrai"

4. LES SYLLOGISMES DANS LE CADRE DE LA QUANTIFICATION.

Nous pouvons facilement férifier la validité des syllogismes classiques du chapitre 3, grâce aux schémas d'énoncé booléens.
Remarquons avant tout que :

'quelque F est G' signifie '$\exists$ FG'
'quelque F n'est pas G' signigie '$\exists F\bar{G}$ '
'tous les F sont G' signifie '$-\exists F\bar{G}$ ', qui équivaut à $(x)(F'+G)x$
'aucun F n'est G' signifie '$-\exists$FG', qui équivaut à $(x)\ (F' + G')x$

Par exemple, démontrons la validité de AAA (1ère figure)

$$(x)\ (T'+G)x \wedge (x)\ (P'+T)x \supset (x)\ (P'+G)x$$
$$\exists TG' \vee \exists PT' \vee -\exists PG'$$
$$\exists PG' \supset \exists (TG' \vee PT')$$

La propriété (d) du paragraphe 3.6.1. nous dit que ce schéma est valide ssi :

$PG' \rightarrow PT'+TG'$
$P'+G+PT'+TG'=1$ ce qui est le cas.

Mais nous pouvons faire bien plus : nous pouvons retrouver les conclusions des syllogismes, si nous leur imposons une forme $(\exists x)$ ou (x).

4.1. La notion d'impliquant primitif

Cherchons les conclusions $c = \exists W$ qui rendent valide le syllogisme EI (1è fig)

$$(x)\ (T'+G')x \wedge (\exists x)\ PTx \supset (\exists x)\ Wx$$
$$\exists TG \vee -\exists PT \vee \exists W$$
$$\exists PT \supset \exists TG \vee \exists W$$

Ce schéma est valide si et seulement si

$PT \rightarrow TG + W$
ou $W' \rightarrow P' + T' + TG$

Autrement dit, toutes les expressions $\exists$ W où W est un impliquant de (P' + T' + G) sont des conclusions valables.
Nous voilà donc amenés à trouver tous les impliquants d'un schéma booléen. Cependant, dès que nous en avons trouvé un, nous en avons trouvé une infinité. En effet, si W implique une fonction booléenne F alors, W multiplié par n'importe quelle fonction booléenne G implique encore F :

si $W \rightarrow F$ $(W' + F = 1)$
alors $WG \rightarrow F$ puisque $W' + G' + F = 1$

La formule (WG) qui contient (W) est un multiple de W (on dit aussi que WG est plus grand, ou, plus long que W).
Les W que nous cherchons sont les plus petits impliquants de F c'est-à-dire les impliquants qui ne sont multiples d'aucun autre impliquant de F : cette classe particulière d'impliquants est celle des impliquants primitifs de F.
Une étude poussée des impliquants primitifs (développant, entre autres, des méthodes de recherche de tous les impliquants primitifs d'une fonction booléenne) est faite au chapitre 1.
Disons ici que si nous voulons trouver tous les impliquants primitifs d'une fonction booléenne mise sous forme normale disjonctive, il nous suffit de l'inverser deux fois; on obtient alors une somme dont chaque élément est un impliquant primitif.
Dans notre exemple :

$W' \rightarrow P' + T' + TG$

en inversant $F = P' + T' + TG$: $\bar{F} = PT(G' + T') = PTG'$

$\bar{\bar{F}} = P' + T' + G$

les impliquants primitifs de P' + T' + TG sont : P', T', G.

Nous pouvons donc prendre W' égale à un de ces impliquants primitifs ou à la somme de un ou plusieurs de ces impliquants.

En prenant W' = P' + G

W = P G',

on trouve la conclusion 0 : $\exists$ P G'

"quelque P ne sont pas G"

4.2. La recherche des conclusions

Ayant deux prémisses qui doivent impliquer une conclusion c :

P1 P2 → c

Le problème est de trouver les conclusions qui rendent valides le conditionnel :

P1 P2 ⊃ c

Dans les syllogismes classiques, les prémisses sont de type :

(1) $(\exists x) \wedge (\exists x)$

(2) $(\exists x) \wedge (x)$

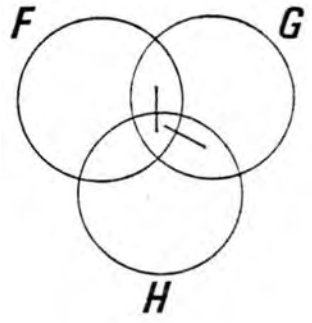
(3) $(x) \wedge (x)$

(4) $(x) \wedge (x) \wedge$ [il existe P' ∨ il existe T' ∨ il existe G']

4.2.1. Illustrons le premier cas par l'exemple :

P1 : il existe des romanciers (F) poètes (G)

P2 : il existe des cinéastes (H) poètes (G)



$\exists FG \wedge \exists HG \supset c$

$-\exists FG \vee -\exists HG \vee c$

en posant $c = (x)\, Wx = -\exists W'$

Ce schéma est valide si et seulement si $-\exists FG$ est valide

ou $-\exists HG$ est valide

ou $-\exists W'$ est valide

Comme les deux premiers ne sont pas valides, si nous voulons un syllogisme valide; il faut que la conclusion $-\exists W'$ soit valide indépendemment des prémisses.

Une telle conclusion serait une tautologie en soi et n'aurait pas grand intérêt.

en posant c = $\exists W$

$\exists FG \wedge \exists HG \supset \exists W$

Ce schéma est valide si et seulement si $FG \rightarrow W$ (a)

ou $HG \rightarrow W$ (b)

En prenant (a) : $W' \rightarrow \overline{FG}$

$W(F' + G') = 0$

égalité qui est vraie si W = FG;

d'où la conclusion $\nexists$ FG

En prenant (b) : $W' \rightarrow \overline{HG}$

$W(H' + G') = 0$,

ce qui est vrai par exemple si W = HG;

d'où la conclusion $\nexists$ HG.

Mais ces deux conclusions ne nous apprennent rien de nouveau : elles répètent les deux prémisses (c'est pourquoi il n'a pas été tenu compte des prémisses $(\exists x) \wedge (\exists x)$ dans la classification des syllogismes).

4.2.2. Pour illustrer le second cas, prenons l'exemple AII (1e fig.)

$(x)\ (T' + G)x \wedge \exists PT \supset c$

$\exists TG' \vee - \exists PT \vee c$

en prenant C = $\exists W$:

$\exists PT \rightarrow \exists(TG' \vee W)$

Ce schéma est valide si et seulement si

$P' + T' + TG' + W = 1$

ou $W' \rightarrow P' + T' + TG'$

en inversant deux fois, nous trouvons les impliquants primitifs de

$(P' + T' + G')$: (P', T', G')

d'où les conclusions : $\nexists$ PTG

$\nexists$ PG, qui est la conclusion T

$\nexists$ TG, qui n'est pas reprise dans la classification des syllogismes.

en prenant c = (x)Wx

$\exists PT \wedge \exists W' \supset \exists TG'$

Ce schéma est valide si et seulement si $PT \rightarrow TG'$

ou $W' \rightarrow TG'$

La dernière implication fournit :

$W = T' + G$

d'où la conclusion $(x)(T' + G)x$ qui répète la première prémisse et n'est donc pas intéressante.

4.2.3. Illustrons le troisième cas par le syllogisme AAA (fig.)

$(x)(T' + G)x \wedge (x)(P' + T)x \supset c$

$\exists TG' \vee \exists PT' \vee c$

en prenant c = (x) Wx

$\exists W' \supset \exists TG' \vee \exists PT'$

Ce schéma est valide si et seulement si $W' \rightarrow TG' + PT'$

Inversons deux fois $(TG' + PT')$ pour trouver ses impliquants primitifs : $(\overline{\overline{TG' + PT'}}) = \overline{T'P' + GP' + GT} = TG' + PG' + PT'$

D'où les conclusions : $(x)(T' + G)x$ (1è prémisse)

$(x)(P' + G)x$ (conclusionA)

$(x)(P' + T)x$ (2d prémisse)

en prenant c = $\exists$ W

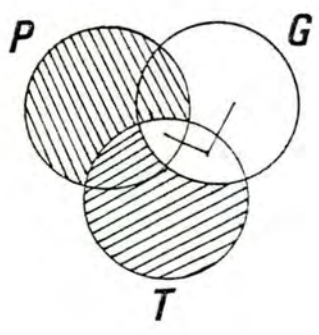
On trouverait suivant le même procédé, la conclusion :

$\exists (T' + G)$

$\exists (P' + G)$

$\exists (P' + T)$

Notons que ceux prémisses universelles ne peuvent fournir une conclusion existentielle que dans un univers non vide où $(x)Fx \rightarrow (\exists x)Fx$



4.2.4. Illustrons le dernier cas par l'exemple AE, 'il existe P' (4è fig)

$$(x)\ (G' + T)x \wedge (x)\ (T' + P')x \wedge \exists P \supset c$$

en prenant $c = \exists W$

Ce schéma est valide si et seulement si $P \rightarrow GT' + TP + W$

$W \rightarrow P' + GT' + TP$

Les impliquants primitifs de $(P + GT' + TP)$ sont :

P', G, et T

En prenant $W' = P' + G$, on trouve la conclusion 0 : $\exists\ PG'$

On peut également trouver : $\exists\ PT'$; $\exists\ G'T'$

Remarquons que toutes les conclusions de syllogismes peuvent aussi être obtenue à partir de la forme prénexe.

Ainsi, pour le syllogisme AA(1èfig.) :

$$(x)\ (T' + G)x \wedge (x)\ (P' + T)x \supset (x)\ Wx$$

$$(\exists y)\ TG'y \vee (\exists z)\ PT'z \vee (x)Wx$$

$$(x)\ (\exists y)\ (\exists z)\ (TG'y \vee PT'z \vee Wx)$$

en éliminant (x) et remplaçant les variables existentielles par x :

$$TG'x \vee PT'x \vee Wx$$

Ce schéma est valide si et seulement si :

$$TG' + PT' + W = 1$$

$$W' \Rightarrow TG' + PT'$$

D'où la solution $W = PG'$ et la conclusion $(x)\ (P' + G)x$

5. RECHERCHE DE CONCLUSIONS A DES ENONCES MONADIQUES

Proposons nous de rechercher des conclusions à des prémisses traduisibles en schéma quantificationnel clos et monadique.

Commençons par un exemple :

P_1 : Les personnes responsables des derniers évènements(g) sont des spécialistes de psychologie expérimentale (h).

P_2 : Si aucun spécialiste de psychologie expérimentale (h) n'est connu de la police (j), alors aucun des anciens patrons du syndicat des trafiquants (f) n'est spécialiste de psychologie expérimentale (h).

$(x)\ (g' + h)x \wedge [(x)\ (h' + g')x \supset (x)\ (f' + h')x] \supset C$

En mettant sous forme normale conjonctive :

$\exists gh' \vee (-\exists hj \wedge \exists fh) \vee C$

$(\exists gh' \vee -\exists hj \vee C)\ (\exists gh' \vee \exists fh \vee C)$

Ce schéma est valide si et seulement si les deux schémas liés par la conjonction sont valides.

1. $\exists gh' \vee -\exists hj \vee C$ [équivalent à $(x)\ (g' + h)x \wedge \exists hj \supset C$]

a) en essayant $C = \exists w$

Le schéma est valide si et seulement si

$$h' + j' + gh' + w = 1$$

d'où on peut prendre $w = hj$

b) en essayant $C = (x)wx$

$\exists hj \wedge \exists w' \supset \exists gh'$

Ce schéma est valide si et seulement si

$$hj \rightarrow gh'$$

ou $w' \rightarrow gh'$

$$w + gh' = 1$$

on peut donc prendre $w = g' + h$

Les deux conclusions trouvées sont :

$\exists hj$ et $(x)\ (g' + h)x$

La deuxième de ces conclusions répète la première prémisse et ne nous apporte rien de nouveau.

2. $\exists gh' \vee \exists fh \vee C$ [équivalent à $(x)(g' + h)(x)(f' + h') \supset C$]

a) en essayant $C = \exists w$

Le schéma est valide si et seulement si

$gh' + fh + w = 1$

$w'(g'f' + g'h' + hf') = 0$

on peut donc prendre $w' = \overline{(g'f' + g'h' + hf')}$

$w' = gh' + gf + fh$

d'où la conclusion $\exists (g' + h)(g' + f')(f' + h')$

et par le théorème 2 du paragraphe 5 on a les 3 conclusions :

$\exists (g' + f')$

$\exists (g' + h)$

$\exists (f' + h')$

b) en essayant $C = (x)w$

$\exists w' \supset \exists gh' \vee \exists fh$

Ce schéma est valide si et seulement si

$w' \rightarrow gh' + fh$

$w'(g'f' + g'h' + hf') = 0$

comme dans le cas a) : $w' = gh' + gf + fh$

$w = (g' + h)(g' + f')(g' + h')$

d'où la conclusion $(x)(g' + h)(g' + f')(f' + h')x$

et par distributivité de (x) sur la conjonction:

on a les 3 conclusions : $(x)(g' + f')x$

$(x)(f' + h')x$

$(x)(g' + h)x$

Les conclusions existentielles tirées en a) ne sont valides que dans un univers de discours non vide : nous les écartons et ne gardons parmi les conclusions de b) que les 2 premières (la troisième étant redondante avec la prémisse P1).

En fin de compte:

le premier schéma nous fournit C_1 : $\exists hj$

le second schéma nous fournit C_2 : $(x)(g' + f')x$

C_3 : $(x)(f' + h')x$

Les conclusions recherchées devant rendre les 2 schémas valides et sachant que : si $(P \vee C_1)$ et $(Q \vee C_2)$ sont tous deux valides,
alors $(P \vee C_1 \vee C_2)$ est valide, ainsi que $(Q \vee C_1 \vee C_2)$
(mais attention, $P \vee C_1 C_2$ et $Q \vee C_1 C_2$ ne sont pas nécessairement valides).

Nous pouvons conclure :

$$C_1 \vee C_2$$
$$C_1 \vee C_3$$

C'est-à-dire :

si $\exists fg$ alors $\exists hj$
si $\exists fh$ alors $\exists hj$

Remarquons que les schémas 1 et 2 ont des prémisses de la forme AI (2è figure) et AE (4è figure).

5.1. Analyse générale

De manière générale, nous considérons le schéma

$$P_1 P_2 \ldots P_n \supset C$$

(où les P_i sont les prémisses données).
Nous le mettons sous forme normale conjonctiveafin d'obtenir une conjonction de disjonction dont chaque disjonction à une des quatre formes ((a) - (d)) vues au paragraphe 3 .
Le schéma sera valide si et seulement si chacune des disjonctions de la conjonction est valide.
Examinons les différents cas possibles pour une disjonction :

1. il s'agit d'une disjonction ne contenant qu'une seule négation de schéma d'existence

$$-\exists f \vee C$$
$$\exists f \supset C$$

En prenant $C = \exists w$

le schéma sera valide si et seulement si

$$f' + w = 1$$
$$w' \Rightarrow f'$$

Nous sommes amenés alors à trouver tous les schémas impliquants le schéma de terme f'.

En prenant C = (x) w

$$-\exists f \vee -\exists w'$$

Ce schéma est valide si et seulement si $(-\exists f)$ ou $(-\exists w')$ est valide indépendemment de la prémisse $(\exists f)$.

Nous ne pouvons donc pas déduire de conclusion de forme universelle.

2. il s'agit d'une disjonction ne contenant aucune négation de schéma d'existence (1 seul schéma d'existence)

$$\exists f \vee \exists g \ldots \vee \exists h \vee C$$
$$\exists (f \vee g \ldots \vee h) \vee C$$

En prenant C = $\exists$ w

le schéma sera valide si et seulement si

$$w' \rightarrow f + g + \ldots + h$$

Cette conclusion n'est valide que dansun univers non vide

En prenant C = (x) w

le schéma sera valide si et seulement si

$$\exists w' \rightarrow \exists f \vee \exists g \ldots \vee \exists h$$
$$w' \rightarrow f + g + \ldots\ldots + h$$

Nous sommes ramenés à chercher tous les schémas de terme w' inpliquant le schéma booléen f + g + ... + h.

3. il s'agit d'une disjonction contenant 1 ou plusieurs négations de schéma d'existence

$$\exists f_1 \ldots - \exists f_i \vee - \exists f_j \ldots \vee \exists f_n \vee C$$

En prenant $C_1 = \exists w_1$

$$\exists f_i \wedge \exists f_j \supset \exists f_1 \vee \ldots \vee \exists f_n \vee \exists w_1$$

Ce schéma est valide si et seulement si

$$f_i \rightarrow \sum_{\substack{k \neq i \\ k \neq j}} f_k + w_1$$

c'est-à-dire $w'_{11} \rightarrow f'_i + \sum_{\substack{h \neq i \\ h \neq j}} f_h$

$$\text{ou } f_j \longrightarrow \sum_{\substack{k \neq i \\ k \neq j}} f_k + w_1$$

$$\text{c'est-à-dire : } w'_{12} \rightarrow f'_j + \sum_{\substack{k \neq i \\ k \neq j}} f_k$$

Sachant que si $(p \vee c)$ et $(p \vee d)$ sont tous deux valides

alors $(p \vee c \vee d)$ et $(p \vee cd)$ sont également valides,

nous pouvons lier les 2 conclusions $(\exists w_{11})$ et $(\exists w_{12})$ par une disjonction ou une conjonction.

Par conséquent, nous retenons les deux conclusions :

$C_{11} = \exists w_{11} \vee \exists w_{12} = \exists (w_{11} \vee w_{12})$

$C_{12} = \exists w_{11} \wedge \exists w_{12}$

En prenant $C_2 = (x) w_2$

$$\exists f_i \wedge \exists f_j \wedge \exists w'_2 \supset \exists f_1 \ldots \vee \exists f_n$$

Ce schéma est valide si et seulement si

$$f_i \rightarrow \sum_{\substack{k \neq i \\ k \neq j}} f_k$$

ou $$f_j \rightarrow \sum_{\substack{k \neq i \\ k \neq j}} f_k$$

ou $$w'_2 \rightarrow \sum_{\substack{k \neq i \\ k \neq j}} f_k$$

Nous pouvons lier les conclusions C_1 et C_2 par une disjonction ou une conjonction, si bien que nous obtenons les conclusions :

$$\exists (w_{11} \vee w_{12}) \vee (x) w_2$$

$$(\exists w_{11} \wedge \exists w_{12}) \vee (x) w_2$$

$$\exists (w_{11} \vee w_{12}) \wedge (x) w_2$$

$$(\exists w_{11} \wedge \exists w_{12}) \wedge (x) w_2$$

De nouveau, la recherche des w_{11}, w_{12} et w_2 se râmène à la recherche des impliquants d'un schéma booléen.

4. il s'agit d'une disjonction ne contenant que des négations de schémas d'existence

$$-\exists f_1 \vee \ldots \vee -\exists f_n \vee C$$

En prenant $C = \exists w$

$$\exists f_1 \wedge \ldots \wedge \exists f_n \supset \exists w$$

Ce schéma est valide si et seulement si

$$w'_1 \rightarrow f'_1$$

$$\text{ou } w'_2 \rightarrow f'_2$$

$$\vdots$$

$$\text{ou } w'_n \rightarrow f'_n$$

Chacune des conclusions ($\exists w_i$) peut être liée aux autres par une disjonction ou une conjonction.

En prenant $C = (x)w$

$$-\exists f_1 \vee \ldots\ldots \vee -\exists f_n \vee -\exists w'$$

Ce schéma n'est valide que si l'un des schémas ($\exists f_i$) ou ($-\exists w'$) l'est. Nous ne pouvons donc trouver de conclusion de forme universelle dépendantes des n prémisses ($\exists f_i$).

Etant donné que la conclusion doit rendre valide chaque disjonction du schéma conjonctif initial, nous devons encore lier par un '$\vee$', les conclusions concernant chacune des disjonctions.

Dans tous les cas, nous sommes réduits à rechercher les impliquants d'une fonction booléenne. Ce problème est longuement analysé au paragraphe 2 du chapitre 1.

CHAPITRE IV.

LES RELATIONS

Les relations sont des termes polyadiques, c'est-à-dire des prédicats à plusieurs places.
Leur introduction est nécessaire pour traiter des inférences telles que :

Tous les cercles sont des figures
Donc, tous ceux qui tracent des cercles, tracent des figures.

En interprétant Hx comme 'x trace un cercle'
Jx comme 'x trace une figure'
Fx comme 'x est un cercle'
Gx comme 'x est une figure'
nous pouvons retranscrire prémisse et conclusion :

$(x)\ (Fx \supset Gx)$ (1)
$(x)\ (Hx \supset Jx)$ (2)

Mais il n'existe aucune connexion de terme entre (1) et (2). Il n'est donc pas possible de tester la validité de l'inférence et a fortiori, il n'est pas possible de déduire (2) de (1). Nous sommes obligés de formuler la relation 'y tracex' par le terme diadique Hyx.
'y trace une figure' se schématise alors en :

$(\exists x)\ (Gx \wedge Hyx)$

et 'y trace un cercle' se schématise :

$(\exists x)\ (Fx \wedge Hyx)$

La conclusion devient

$(y)\ \left[(\exists x)\ (Fx \wedge Hyx) \supset (\exists x)\ (Gx \wedge Hyx)\right]$

1. Test de validité

Contrairement aux énoncés monadiques :

> Il n'est pas toujours possible de mettre un énoncé polyadique sous forme prénexe, où tous les quantificateurs universels sont en tête.

Mais si cela est faisable, nous pouvons tester sa validité suivant la procédure décrite au paragraphe 3.5. du chapitre 3.
Testons la validité de l'inférence précédente :

$(y)\ (z)\ (\exists x)\ (\exists w)\ \left[F\bar{G}\ x \vee \bar{F}z \vee \bar{H}yz \vee (Gw \wedge Hyw)\right]$

3. Une relation R est totalement réflexive ssi

$(x)\ Rxx$

Elle est réflexive ssi

$(x)(y)\ [Rxy \supset (Rxx \wedge Ryy)]$

Elle est irréflexive ssi

$(x)\ \overline{R}xx$

3. Recherche d'une conclusion

Pour autant qu'un énoncé puisse se mettre sous la forme d'un schéma existentiel pur (forme prénexe sans quantificateur universel), nous pouvons chercher des conclusions en nous basant sur la procédure de test.

Par exemple : représentons :

'x est père de y' par Pxy

'x est grand-père de y' par GPxy

Nous pouvons émettre 1 contrainte sur ces relations :

$(x)(y)(z)\ (Pxy \wedge Pyz \supset GPxz)$

Cherchons alors des conclusions à :

'André est père de Bernard' :Pab

'Bernard est père de Charles' :Pbc

$(x)(y)(z)\ (Pxy\ Pyz \supset GPxz) \wedge Pab \wedge Pbc \supset C$

$(\exists x)(\exists y)(\exists z)\ [Pxy\ Pyz\ \overline{GP}xz \vee \overline{P}ab \vee \overline{P}bc \vee C]$

Il y a 3 variables existentielles, 3/ variables libres et donc 27 substitutions possibles.

l'une de celles-ci (a x,b y,c z) donne le schéma :

$Pab\ Pbc\ \overline{GP}ac \vee \overline{P}ab \vee \overline{P}bc \vee C$

Pour rendre ce schéma valide, il suffit de prendre

$\overline{C} = Pab\ Pbc\ \overline{GP}ac \vee \overline{P}ab \vee \overline{P}bc$

$C = (\overline{P}ab \vee \overline{P}bc \vee GPac)\ Pab\ Pbc$

$C = Pab\ Pbc\ GPac$

d'où la conclusion :

'André est grand-père de Charles'

4. Les relations univoques

Ce paragraphe concerne la résolution de problèmes logiques faisant intervenir des relations univoques.
Une relation Rx,y,..,z est univoque si et seulement si pour chaque valeur d'un argument (x,y,...ou z) il n'y a qu'une seule valeur permise pour chacun des autres arguments.
A partir d'une même application, nous allons, à l'aide de 4 méthodes différentes, définir des approches spécifiques pour la recherche d'une solution. Ces quatre méthodes se basent chacune sur une démarche logique commune que nous détaillerons mais dont les procédures de traitement ne s'appliqueront que dans des domaines d'applications différentes.
Ainsi, le premier procédé n'est valable que pour des problèmes simples et les suivants pour des problèmes de plus en plus complexes.
On aboutira ainsi à la 4ème méthode pour des applications qui imposeront d'adopter une technique plus générale, en l'occurence de nature binaire, et surtout plus performante.

Enoncé de l'application

Trois personnes se prénommant diféremment exercent des métiers distincts et résident dans des localités différentes (Paris, Lyon, méditerranée). Il s'agit de les identifier au moyen des prémisses suivantes :

1. Jean habite Paris.
2. Le mécanicien habite Lyon
3. Pierre qui bat le chauffeur au billard a rencontré le serre-frein qui lui a avoué ne pas connaître Paris

Quelle est la profession de Roger ?

Démarche commune d'approche

En premier lieu, nous symboliserons les composants de la classe des personnes au moyen des lettres J, P, R; ceux de la classe des fonctions par F_m, F_s, F_c et ceux de la classe des résidences par R_p, R_l, R_m.
En second lieu, nous rechercherons et symboliserons les relations élémentaires. c'est-à-dire ne contenant qu'une relation, si cela est possible.
Nous transposons ainsi l'énoncé en :

$$S_1 \begin{cases} (1)\ H_{JR_p} & H_{xy} = \text{'x habite y'} \\ (2)\ H_{F_m R_1} & \\ (3a)\ N_{P\bar{F}_s\bar{F}_c} & N_{x\bar{y}\bar{z}} = \text{'x n'est ni y, ni z'} \\ & \text{on a } \bar{F}_s\ \bar{F}_c = F_m \\ (3b)\ \bar{H}_{F_s R_p} & \end{cases}$$

4.1. 1ère méthode de résolution - Algèbre logique

Nous pouvons traduire les 4 relations du système S_1 sous la forme "homologique" et obtenir :

$$S_2 \begin{cases} JR_p + \bar{J}\bar{R}_p & (1) \\ F_m R_1 + \bar{F}_m \bar{R}_1 & (2) \\ PF_m + \bar{P}\bar{F}_m & (3a) \\ F_s\bar{R}_p + \bar{F}_s R_p & (3b) \end{cases}$$

La solution s'obtiendra en effectuant le produit logique des 4 prémisses. Le résultat de ce produit global contiendra donc 16 composants qui devront se réduire à trois, représentant les relations entre personnes, fonctions, et résidences. Des réductions et simplifications sont donc à opérer. Celles-ci sont possibles grâce aux égalités suivantes (qui existent pour chaque classe : personnes, résidences et fonctions).

$$J\bar{P} = 0;\ \bar{J} = P + R;\ \bar{J}\bar{P} = R;\ \bar{J}P = \bar{J}P\bar{R} = P \quad \text{etc ...}$$

Le produit de (1) avec (2) donne:

$$J\bar{F}_m R_p + \bar{J}F_m R_1 + \bar{J}\bar{F}_m R_m \qquad (4)$$

Le produit logique (4) avec (3a) donne :

$$PF_m R_1 + J\bar{F}_m R_p + R\bar{F}_m R_m \qquad (5)$$

d'où l'élément de réponse : $PF_m R_1$.

Et enfin, le produit de (5) avec (3b) donne

$$RF_s R_m + JF_c R_p$$

indiquant que Roger est serre-frein et habite la région méditéranéenne.

4.2. 2ème méthode : tabulaire classique

La table de décision classique devient vite inutilisable dans le cas de problèmes de logique. Elles mettent en regard, les décisions (solutions ou non) pour toutes les combinaisons des conditions primaires, dans notre cas, les combinaisons des trois variables de nature binaire.
On devrait donc élaborer pour cet exercice simple, une table à $3^3 = 27$ colonnes et pour chacune, analyser l'énoncé pour déterminer les solutions possibles.
Comme on le fait dans ce cas là pour les tables de décisions, il y a intérêt à travailler avec plusieurs tables à double entrée, c'est-à-dire, dans notre cas, à 3 tables (P,F), (P, R) et (R, F).
Dans les 27 cases de ces tables, on met 1 ou 0 selon que la décision est compatible avec l'énoncé.

1

	R_p	R_l	R_m
J	1	0	0
P	0		
R	0		

2

	F_m	F_s	F_c
R_p	0	0	
R_l	1	0	0
R_m	0	1	

3b

3a

	F_m	F_s	F_c
J	0		
P	1	0	0
R	0		

J n'est compatible qu'avec R_p qui n'est compatible qu'avec F_c.
Donc, R n'est compatible qu'avec F_s, lui-même compatible avec R_m.

	F_m	F_s	F_c
J	0	0	1
P	1	0	0
R	0	1	0

	R_p	R_l	R_m
J	1	0	0
P	0	1	0
R	0	0	1

	F_m	F_s	F_c
R_p	0	0	1
R_l	1	0	0
R_m	0	1	0

4.3. 3ème méthode : tabulaire avec tassement

Comme pour la méthode précédente, on ne transpose plus l'énoncé en équations logiques pour en exprimer les relations, mais on élabore une table où chacune des classes occupe une colonne de la table et où les éléments composant chaque classe se répartissent dans la colonne correspondante. Cette répartition tient compte des relations qui les unissent; chaque relation occupe donc une ligne du tableau.
On devra ensuite opérer un tassement des lignes du tableau pour en ramener le nombre à celui des relations unissant les éléments des classes, en l'ocurrence 3.
Les principes généraux qu'il y a lieu d'appliquer sont présenté ci-après, pour des cas d'une certaine complexité. Toutefois, dans notre application simple, une version réduite est développée dans l'ordre suivant :

1. transposer les relations dans une table (P, R, F), y occupant chacune une ligne
2. Fusionner les lignes qui ont un mot-lien commun, soit (2) et (3a) avec F_m
3. Lever les ambiguités. Ainsi, R_1 étant lié à P, ne peut l'être à R, d'où suppression de R_1*
4. Combler les cases vides restantes. Ainsi, F_m et F_s étant les professions respectives de P et de R, F_c ne pourra être que celle de J.

classe	P	R	F
	J	R_p	
1		R_1	F_m
2	P		F_m
3a	R	R_1+ R_m	F_s
3b	J	R_p	
1	P	R_1	F_m
2-3a	R	R_1+ R_m	F_s
3b	J	R_p	F_c
	P	R_1	F_m
	R	R_m	F_s

Principe général du tassement

L'application de ce principe permettra de traiter une table de relations pour obtenir au mieux la table des solutions, au pire une table intermédiaire.

Pour ce faire, on examinera successivement les éléments de chaque classe (colonne) par rapport à ceux des autres classes.

En vertu des relations qui existent entre les éléments d'une colonne et ceux d'une autre, on pourra introduire dans les cases vides la négation des éléments qui ne correspondent pas aux relations parfaitement définies.

On procédera comme suit :

Considérons deux colonnes X et Y d'une table de relations.

Si l'élément X_i de la colonne X est associé par relation à l'élément Y_j de la colonne Y, pour tous les éléments de X qui sont différents de X_i, on pourra introduire dans la colonne Y, l'élément $\bar{Y}_j$.

$\bar{Y}_j$ risque de rencontrer

- une case vide et sera enregistré sous sa forme $\bar{Y}_j$
- un élément Y_k qui restera inchangé puisque $Y_k \bar{Y}_j = Y_k$
- un élément $\bar{Y}_e$ qui deviendra $\bar{Y}_e \bar{Y}_j = \sum$ (Y complémentaires)

Pour illustrer cette proposition, considérons le tableau :

W	X	Y	Z
W_a	X_e	–	–
W_b	X_f	–	Z_m
–	X_g	Y_i	–
W_c	–	Y_j	Z_n

Pour tous les W	Placer dans case correspondante de			
	W	X	Y	Z
$\neq W_a$	-	$\bar{X}_e$	-	-
$\neq W_b$	-	$\bar{X}_f$	-	$\bar{Z}_m$
$\neq W_c$	-	-	$\bar{Y}_j$	$\bar{Z}_m$

Pour tous les X				
$\neq X_e$	$\bar{W}_a$	-	-	-
$\neq X_f$	$\bar{W}_b$	-	-	$\bar{Z}_m$
$\neq X_g$	-	-	$\bar{Y}_i$	-

Pour tous les Y				
$\neq Y_i$	-	$\bar{X}_g$	-	-
$\neq Y_j$	$\bar{W}_a$	-	-	$\bar{Z}_n$

Pour tous les Z				
$\neq Z_m$	$\bar{W}_b$	$\bar{X}_f$	-	-
$\neq Z_n$	$\bar{W}_c$	-	$\bar{Y}_j$	-

Le tableau initial se transforme ainsi en :

W	X	Y	Z
W_a	X_e	$\bar{Y}_i\,\bar{Y}_j$	$\bar{Z}_n\bar{Z}_m$
W_b	X_f	$\bar{Y}_i\,\bar{Y}_j$	Z_m
$\bar{W}_a\,\bar{W}_b\,\bar{W}_c$	X_g	Y_i	$\bar{Z}_n\bar{Z}_m$
W_c	$\bar{X}_e\,\bar{X}_f\,\bar{X}_g$	Y_j	Z_n

Cette méthode séduisante pour certains problèmes de logique atteint rapidement ses limites lorsque la barre des difficultés s'élève.
Ceci résulte principalement de l'apparition dans les éléments des relations, d'expressions algébriques plus ou moins complexes qu'il faut traiter sous toutes les formes logiques (produit, somme, négation) et l'on sait que ces opérations sont pleines de pièges.
Cette observation milite en faveur de l'adoption de la quatrième méthode qui en vertu de sa nature et de ses moyens, n'a pratiquement pas de limites.

4.4. Méthode principale : opérations logiques binaires

Toutes les opérations logiques réalisées dans les méthodes précédentes pour un cas simple d'espèce peuvent être généralisées à des cas complexes pour lesquels le nombre de relations est non seulement important mais où la nature de celles-ci est multiple, du genre

$R_{ab} \wedge [H(R_{cd} \vee F_{de})]$

La méthode nouvelle est caractérisée par les faits suivants :

1. Découper l'énoncé en un ensemble de relations.
   Ce faisant, on y répertorie les différentes classes (personnes, objets, attributs, compléments...) et dans chacune de celles-ci les différents éléments.

2. Coder les relations dans un langage commun c'est-à-dire binaire pour tous les éléments de n'importe quelle classe.
   Une relation est exprimée par un vecteur où chaque champ est dédicacé à une classe et où la position des champs dans le vecteur définit implicitement la relation.
   Un champ contient autant de bits qu'il y a d'éléments dans la classe qu'il représente.

3. En ce qui concerne les champs :
   a) ils peuvent être de types différents :
      - simples : et complètement définis s'ils comptent un seul bit 1; ils définissent alors sans ambiguité un élément de la classe, selon la position du '1' dans le champ.
      - multiples : et incomplètement définis s'ils comptent plusieurs bits 1 et au moins un 0, ou complètement indéfinis s'il ne comptent que des 1.
        Un champ contenant plusieurs 1 représente plusieurs éléments reliés entre eux par l'opérateur "ou".

b) Ils peuvent être de catégories différentes :

- les champs clés, toujours présents, seront choisis pour servir de référence (comme un indicatif de record pour un fichier, tel que n° article, nom, etc...). La solution globale comportera autant de vecteurs qu'il a d'éléments (de mots clés) représentés par le champ clé.
- les champs liens assurent la liaison entre les vecteurs. Ils déterminent la position (dans l'espace, le temps, la hiérarchie,...) d'une personne, objet, attribut,... par rapport aux autres.

4. En ce qui concerne les vecteurs

a) ils peuvent être élémentaires : c'est-à-dire être seuls et représenter une proposition (par exemple :'Jean habite Paris'.

b) ils peuvent être composés : ils sont alors plusieurs, groupés et reliés entre eux par l'opérateur 'et' pour représenter 1 proposition composée (par exemple : Jean habite Paris et Roger est mécanicien).

5. On systématisera les opérations logiques et notamment les produits soit manuellement, soit par programmation si le traitement est trop laborieux sinon impossible par les moyens habituels.
Quel que soit l'article utilisé, la méthode systématique consistera :

a) à transformer le vecteur d'1 relation contenant des champs multiples en plusieurs vecteurs ne contenant que des champs simples.
Les vecteurs ainsi complètement définis sont reliés entre eux par un 'ou'. Cette transformation se fait aussi bien pour les vecteurs élémentaires que pour les vecteurs composés.

b) à effectuer les produits logiques entre les vecteurs pris deux à deux dans chacune des relations.

c) à examiner les résultats de ces produits.
Si le produit = 0, tous les champ sont nuls dans le résultat.
Les 2 vecteurs multipliés fournissent deux éléments d'une solution possible.

1. Si le produit ≠ 0 et au moins un champ (mais pas tous) est différent de 0. Dans ce cas, aucun des vecteurs d'origine ne peut appartenir à une solution possible.
2. Si le produit ≠ 0 et tous les champs sont différents de 0. Comme ils sont complètement définis, les deux vecteurs d'origine sont identiques et constituent un élément d'une solution possible.

d) on obtiendra finalement la solution globale définitive si le nombre des solitions individuelles et différentes est égal au nombre de mots clés dénombrés au départ.

L'algorithme général de résolution est décrit point 4.4.4.

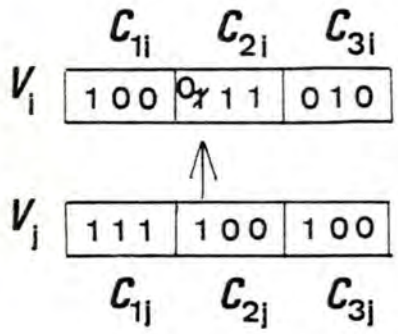
4.4.1 Simplifications

Il s'agit de simplifications que l'on apporte à un certain nombre de vecteurs, immédiatemment après leur création qui a été effectuée lors du découpage de l'énoncé en relation , et avant leur expansion en vecteurs simples et complètement définis. Cette simplification peut alléger considérablement la procédure de la méthode systématique et à la limite l'éviter en aboutissant à la solution finale.
Pour l'obtenir, on effectuera successivement entre certains vecteurs de relation, la procédure suivante :

Lorsque deux vecteurs sont tels que :

- l'un des V_j possède au moins 2 champs complètement définis (ici C_{2j} et C_{3j});
- l'autre des vecteurs, V_i, possède au moins un champ complètement défini dans la même zone que celui de V_j qui est complètement défini,

alors on pourra mieux ou tout à fait définir un autre champ (C_{2i}) incomplètement défini de V_i en remplaçant un bit 1 par un bit 0 (ce bit étant de même poids que celui de C_{2j}).



De la même façon, le champ C_{1j} pourrait devenir 011 étant donné que $(C_{1i}) = 100$.

Ces opérations étant réalisées de proche en proche, l'expansion en vecteurs simples sera plus réduite, comme les opérations logiques systématiques.

Voici l'exemple d'un exercice où les simplifications amènent directement à la solution :

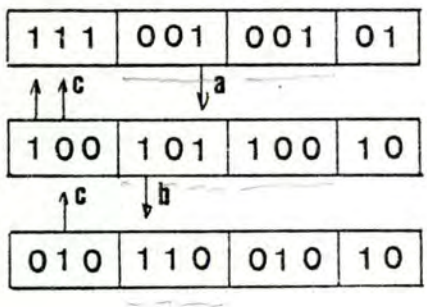
Lebrun, Lenoir et Leblanc travaillent ensemble dans une même entreprise. Ils sont : comptable, magasinier et représentant, mais peut être dans un ordre différent.
Le représentant, qui est célibataire, est le plus petit des trois;
Lebrun, qui est le gendre de Lenoir, est plus grand que le magasinier.
Quel est le métier de chacun ?

a) Recherche de classes

	Personnes	Professions	Tailles	Etat civil
Leblanc	0 0 1	R 0 0 1	P 0 0 1	m 1 0
Lenoir	0 1 0	M 0 1 0	M 0 1 0	$\bar{m}$ 0 1
Lebrun	1 0 0	C 1 0 0	G 1 0 0	

Le champ clé est delui des personnes (et les mots clés sont : Lebrun, Lenoir, Leblanc).
Les champs liens, sont : Personnes, Professions, Tailles.

b) Transposition de l'énoncé



c) Simplifications (opérations a,b,c)

001	001	001	01
100	100	100	10
010	010	010	10

4.4.2. Solution de l'application

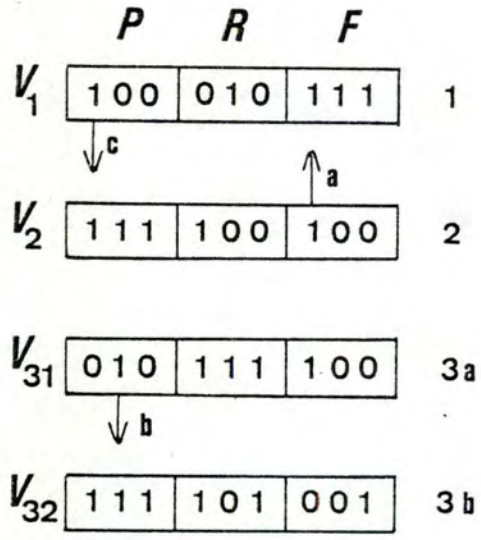
a) Codage des relations

Il s'agit de coder le système S_1

Nous obtenons :

Personnes	J(1 0 0)	P(0 1 0)	R(0 0 1)
Résidences	L(1 0 0)	P(0 1 0)	M(0 0 1)
Fonctions	M(1 0 0)	C(0 1 0)	SF(0 0 1)

(Tous les vecteurs sont de type'élémentaires')



b) Simplifications (opérations a,b et c)

100	010	011

011	100	100

010	111	100

101	101	001

c) Expansion des champs_multiples_

V_{11}	100	010	010	
V_{12}	100	010	001	ou
V_{21}	010	100	100	
V_{22}	001	100	100	+
V_{311}	010	100	100	
V_{312}	010	010	100	+
V_{313}	010	001	100	+
V_{321}	100	100	001	
V_{322}	100	001	001	+
V_{323}	001	100	001	+
V_{324}	001	001	001	+

$V_{11} \cdot V_{321/2/3/4} = 0$ $\quad V_{21} \cdot$

$S_1 \cdot S_3' = 0$ + $S_2 \cdot S_3 = 0$

$(S_1 + S_2)\, S_3 = 0$ ⇒ $S_3 \neq S_1$ ou S_2

On distingue quatre groupes de relations; chaque groupe contient des vecteurs élémentaires reliés par un 'ou'.

1er groupe on prend V_{11}

2è groupe $V_{11} \cdot V_{21} = 0$ on retient les vecteurs V_{11} et V_{21}

La solution doit contenir 3 vecteurs (nombre de mots clés)

Le troisième vecteur doit être tel que son produit avec $V_{11} = 0$, ainsi que sont produit avec V_{21}.

Autant dire que son produit avec $X = V_{11} + V_{21}$ doit être nul.

X			
	110	110	110

On passe au groupe suivant

3è groupe $X \cdot V_{311} \neq 0$

$X \cdot V_{311} = V_{21}$ ce vecteur est déjà retenu

on passe au groupe suivant

4è groupe $X \cdot V_{321} \neq 0$

$X \cdot V_{322} \neq 0$

$X \cdot V_{323} \neq 0$

$X \cdot V_{324} = 0$ on retient le vecteur V_{324}

Trois vecteurs ont été retenus. Ils forment la solution :

100	010	010

010	100	100

001	001	001

Si la solution n'avait pas été trouvée après observation des vecteurs du dernier groupe, il aurrait fallu remonter au dernier vecteur retenu (soit, ici, V_{21}), le rejeter, et continuer la procédure avec le vecteur suivant du même groupe (V_{22}). Si ce groupe ne contient plus de vecteur, il faut remonter à l'avant dernier vecteur retenu, le rejeter, prendre le suivant du même groupe, continuer la procédure et ainsi de suite...

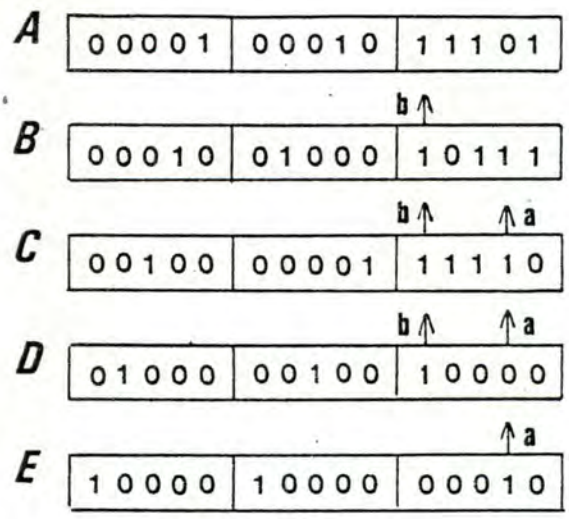
4.4.3. Exercice

André, Bernard, Claude, Didier et Etienne ont construit leurs villas autour d'une petite plage de l'Atlantique.
D'un commun accord, ils ont décidé que chacun donnerait à sa villa le nom de la fille de l'un de ses quatre amis. Il se trouve en effet que chacun a une fille : Anne, Belle, Cécile, Dona et Eve (mais pas nécessairement dans cet ordre).
Pour éviter que deux villas n'aient le même nom, ils se sont réunis pour faire leur choix. Il en a résulté :
Claude et Bernard qui désiraient tous les deux appeler leur villa Dona, ont tiré au sort. Bernard a gagné. Claude a nommé sa villa Anne.
André a nommé sa villa Belle.
Le père d'Eve étant absent, c'est Etienne qui lui a téléphoné que sa villa se nommait Cécile.
Le père de Belle a nommé sa villa Eve.
Quelle est la fille et quelle est la villa de chacun ?

a) Recherche des classes et éléments

	Père	Villa(nom)		Fille(nom)	
André (A)	0 0 0 0 1	Anne	0 0 0 0 1	A	0 0 0 0 1
Bernard (B)	0 0 0 1 0	Belle	0 0 0 1 0	B	0 0 0 1 0
Claude (C)	0 0 1 0 0	Cécile	0 0 1 0 0	C	0 0 1 0 0
Didier (D)	0 1 0 0 0	Dona	0 1 0 0 0	D	0 1 0 0 0
Etienne (E)	1 0 0 0 0	Eve	1 0 0 0 0	E	1 0 0 0 0

b) Codage des relations



c) Simplifications (opérations a et b)

E	10000	10000	00010
D	01000	00100	10000
C	00100	00001	01100
B	00010	01000	00101
A	00001	00010	01101

d) Expansion des champs multiples

E	10000	10000	00010
D	01000	00100	10000
C_1	00100	00001	01000
C_2	00100	00001	00100
B_1	00010	01000	00100
B_2	00010	01000	00001
A_1	00001	00010	01000
A_2	00001	00010	00100
A_3	00001	00010	00001

e) Résolution

On distingue 5 groupes de vecteurs élémentaires

1er groupe : on garde E

2è groupe : E . D = 0

on garde D

X = E + D

X

11000	10100	10010

3è groupe : X . C_1 = 0

on garde C_1

X = X + C_1

X

11100	10101	11010

4è groupe : X . B_1 = 0

on garde B_1

X = X + B_1

X

11110	11101	11110

5è groupe : X . $A_1 \neq 0$

X . $A_2 \neq 0$

X . A_3 = 0

on garde A_3

d'où la solution :

10000	10000	00010	E
01000	00100	10000	D
00100	00001	01000	C_1
00010	01000	00100	B_1
00001	00010	00001	A_3

Cette solution n'est pas unique. En reprenant la procédure à partir de B_2 :

4è groupe : $X \cdot B_2 = 0$

on garde B_2

$X = X + B_2$

X			
	11110	11101	11011

5è groupe : $X \cdot A_1 \neq 0$

$X \cdot A_2 = 0$ garder A_2

d'où la solution :

	10000	10000	00010
	01000	00100	10000
	00100	00001	01000
	00010	01000	00001
	00001	00010	00100

E

D

C_1

B_2

A_2

4.4.4. Algorithme de résolution

Rappelons qu'après expansion des champs, les vecteurs ne contiennent que des champs simples, et peuvent être :

- élémentaires
- composés : il s'agit alors d'un ensemble de vecteurs reliés par un 'et'. Il faudra soit les accepter tous dans la solution, soit les rejeter tous.

Les champs qui ne sont pas de champs liens sont en général placés en bout des vecteurs. Dans l'algorithme ci-dessous, nous considérons des vecteurs uniquement constitués de champs liens (dont un champ clé). Les vecteurs appartiennent à des groupes. Les vecteurs d'un même groupe sont reliés entre eux par un 'ou'.

Chaque groupe de vecteurs représente une proposition de l'énoncé, supposée vraie. Par conséquent, un groupe de vecteur contient au moins un élément de la réponse (sinon plusieurs);
tous les vecteurs seront multipliés par un vecteur X. X contient la somme booléenne des éléments de solution retenus. Il est initialisé à 0 (ses champs ne contiennent que des bits nuls) et mis à jour au fur et à mesure que l'on retient des vecteurs dans la solution.
Soit N, le nombre de vecteurs retenus dans un groupe :

```
Pour tout groupe de vecteurs
    N = 0
    Pour tout vecteur Vi du groupe
        si produit(x, Vi) = 0   retenir Vi comme élément de la solution;
                                N = N+1
        si produit(x, Vi) ≠ 0   soit que Vi est un élément déjà retenu
                                (faire N=N+1)
                                soit que Vi n'est pas un élément à retenir
    Si N = 0, le groupe ne contient aucun élément de la réponse; on fait donc fausse route : il faut éliminer de la solution le dernier vecteur retenu et recommencer à partir du suivant.
```

On écrira les groupes de vecteurs contenant un seul vecteur (élémentaire ou composé) dans un tableau "vectseul". Ces vecteurs sont obligatoirement des éléments de la solution. Ils seront séparés au sein du tableau "vectseul" par un caractère de bits 1 ('\1'). A la suite du dernier vecteur, mettra le caractère nul ('\0').

Le tableau 'vectnonseul' contiendra les autres groupes de vecteurs. Pour des raisons d'efficacité, on écrira d'abord dans ce tableau, les groupes de vecteurs composés (ceux-ci contenant plusieurs vecteurs, ils peuvent nous conduire plus vite à la solution, si on les considère en premier lieu).

Au sein d'un groupe, les vecteurs seront séparés par le caractère nul '\0'.

Les groupes seront séparés par le caractère '\1' (= 127 en décimal, dans le code ASCII). Le tableau "solution" contient les éléments retenus sous forme de pointeurs, chacun pointant vers un vecteur de "vectseul" ou de "vectnonseul". Le pointeur "fin" pointe vers le premier élément libre du tableau "solution". Le pointeur "vect" pointe vers le début du vecteur V_i courant.

La constante "champ" représente le nombre de caractères utilisés pour coder un vecteur.

La constante "nbresol" représente le nombre de mots clés, et donc le nombre de vecteurs dans la solution.

1ère étape : La première chose à faire est de mettre dans "solution" les pointeurs pointant vers chaque vecteur de "vectseul".
Faire X = somme booléenne des vecteurs de "vectseul"
(les vecteurs constituant les vecteurs composés sont également sommés).
par exemple :

vectseul

0100	1000
1	
0010	0010
0001	0100
1	
π = 0	

et

0111	1110

X

```
for (i=0; i!=champ; i++)
                x[i] != '\0';
solution[0] = NULL;
fin = &solution[i]; (fin = adresse de solution[i]/
vect = &vectseul[0];
while (*vect!='\0')
        {
        *fin++ = vect
        for(i=0; i!=champ; i++)
                x[i] = x[i] | *vect++
        if (*vect = 127) vect++
        }
```

2è étape : Il s'agit à présent de comparer entre eux tous les vecteurs de "vectnonseul" et de ne garder que ceux qui sont compatibles entre eux (dont le produit donne un vecteur dont tous les champs sont nuls).

```
vect = & vectnonseul [0]
N = 0
while ( ( fin ! = solution + nbresol +1) & & (fin! = solution))
    if (* vect = = '0') vect ++ ;
    if (* vect = = 127) | N = 0 ;
                        | vect ++ ;
    v = vect ;
    r = test (x, vect) ;
    if (r ! = - 1)
        | N ++ ;
        | if (r! = 0) | for (i = 0; i ! = champ; i++)
        |             |     * fin ++ = v++
    else
        | if ( ( * vect = = - 127) & & ( N == 0))
        |     | while ( (*vect = = - 127) & & (fin! = solution))
        |     |     | for (i = 0; i ! = champ; i ++)
        |     |     |     T [i] = '0';
        |     |     | fin = fin - 1;
        |     |     | vect = * fin;
        |     |     | while ( ( * vect ! = '0') || (* vect! = 127))
        |     |     |     | for (i = 0; i ! = champ; i++)
        |     |     |     |     . T [i] = * vect++ | T [i]
```

```
                    for (i=0; i!=champ; i++)
                        T[i] = ~T[i]
                        x[i] = x[i] && T[i]
    if (fin == solution + nbresol+1)
                afficher solution
    if (fin == solution)
                Printf ("il manque des relations
                        pour obtenir une solution
                        complete")
```

test (X, vect)

Cette procédure compare le vecteur vect, candidat comme élément à la solution, avec le vecteur X, somme booléenne de tous les éléments retenus dans la solution.

- Si produit (X, vect) = un vecteur dont tous les champs sont nuls, alors vect est un vecteur à retenir.
  "test" retourne alors le nombre de vecteurs élémentaires à retenir, soit 1 si vect est un vecteur élémentaire,
  soit le nbre de vecteurs élémentaires constituant vect si celui-ci est un vecteur composé.
- Si produit (X, vect) ≠ un vecteur dont tous les champs sont nuls mais vect a déjà été retenu, alors "test" retourne un entier = 0.
- Si produit (X, vect) ≠ un vecteur dont tous les champs sont nuls et vect n'a pas encore été retenu, alors "test" retourne un entier = -1.
- En outre, si vect est un vecteur à retenir, X est mis à jour.

```
for (i=0; i!=champ; i++)
        | T[i] = 127;
          Y[i] = '\0';
h = 0;
v = vect;
while ((*v!='\0') || (*v!=127))
        | h++;
        | for (i=; i!=champ; i++)
        |     T[i] = T[i] & *vect++;
for (i=0; i!=champ; i++)
        R[i] = T[i] & X[i]
c = comparaison (R,Y);
if (c==0)
        | for (i=0; i!=champ; i++)
        |       X[i] = X[i] | T[i];
if (c!=0)
        | p = &solution[0];
        | l = -1
        | while ((*p!=fin) && (l==-1))
        |               | p++;
        |               | l = comparaison (*p, v);
        | if (l==0)
        |       h=0
        |   else
        |       h=-1
return (h)
```

comparaison (a,b)

Cette procédure compare deux vecteurs a et b.
Si ceux-ci sont égaux, elle retourne 0. Sinon, elle retourne -1

```
k = 0;
i = 0;
while (a[i] != champ) && (k == 0))
        if (*a++ != *b++)
                k = -1;
        i++;
return (k)
```

L'écriture des vecteurs se fait suivant la syntaxe :

<texte> → <premisse> $
<texte> → <premisse> ; <texte>
<premisse> → > <relation seule>
<premisse> → <relation non seule>
<relation non seule> → <vecteur>
<relation non seule> → <vecteur> + <relation non seule>
<relation seule> → <vecteur>
<vecteur> → <vecteur simple>
<vecteur> → <vecteur simple> , <vecteur>
<vecteur simple> → <entier>
<vecteur simple> → <entier> sp <vect simple>

un <entier> est un des nombres (de 0 à 127) du code ASCII, utilisé pour coder les caractères composant un vecteur.

C O N C L U S I O N

La logique des propositions et des prédicats sans relation trouvent leur aboutissement dans les programmes "compile .c" et "interro .c" décrits en annexe.

Une modification importante peut facilement leur être apportée : on pourrait imaginer un langage de programmation permettant de définir des classes d'objets, des attributs définissant ces classes, des objets (éléments des classes) et leurs valeurs d'attributs.
Les prémisses seraient du type "implication" et lieraient entre eux les objets d'une même classe ou de classes distinctes.
Un objet ne serait plus représenté par un doublet de bits mais par un vecteur contenant autant de champs qu'il y a d'attributs caractérisant la classe de l'objet. Un champ contiendrait autant de bits qu'il y a de valeurs possibles pour l'attribut qu'il symbolise.
En dehors de cela, les principes de résolution seraient rigoureusement identiques.

Ainsi améliorés, les programmes "compile .c" et "interro .c" permettraient par exemple, de programmer efficacement les tables de décision polyvalentes.

Quant au chapitre des relations, il est loin d'être complet.
Mais il constitue une base solide pour entreprendre une tâche grandiose : la conception d'un langage de programmation et d'interrogation aptes à résoudre des exercices dans le cadre de la logique des prédicats du second ordre (ou des ordres supérieurs) et avec relations.

Bibliographie

1. Méthodes de logique - W. QUINE
(Armand COLLIN - collection U)

2. Algèbre de boole - J. KUNTZMANN
(Dunod)

3.Monographie des treillis et algèbre de boole - M. CARVALLO
(Gauthiers - Villars)

4. Logique des circuits câblés et des programmes enregistrés - J. BRUNIN
(Travaux de l'Institut d'Informatique - n° 1)

5. Logique binaire et commutation - J. BRUNIN
(Dunod)

6. Comprendre la logique moderne - F. CHENIQUE
(Dunod)

7. Introduction à la logique - A TARSKI
(Gauthiers - Villars)

TABLE DES MATIERES

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX

(NAMUR)

INSTITUT D'INFORMATIQUE

Annexe

Logique du raisonnement

APPLICATION A LA LOGIQUE DES PREDICATS SANS RELATION

Mémoire présenté par
Freddy LORGE
en vue de l'obtention du titre de
LICENCIE ET MAITRE EN INFORMATIQUE
Promoteur : J. BRUNIN
Année académique : 1984/1985

INTRODUCTION

Cette annexe décrit la façon dont a été imaginé et conçu un logiciel capable de trouver toutes les conclusions relatives à un nombre quelconque de prémisses, dont la structure est une généralisation des 4 formes A, E, I, O.

Plus précisément, les prémisses envisagées doivent avoir une des formes générales :

(x) (forme normale disjonctive) (x)

($\exists$ x) (conjonction) (x)

Ce logiciel est en fait composé de 2 programmes :

- un compilateur ("compile.c") qui analyse et code les prémisses introduites par l'utilisateur via un langage de programmation.
- un programme d'exécution ("interro.c") qui fournit les conclusions relatives à des termes précis, introduits par l'utilisateur via un langage d'interrogation.

1. Principe de résolution

1.1. Recherche des conclusions pour prémisses universelles

Les questions posées par l'utilisateur auront la forme générale :

forme normale disjonctive $\longrightarrow$?

Soit un texte constitué des 2 prémisses :

$(x)\ (fnd_1)\ x$

$(x)\ (fnd_2)\ x$

et soit la question :

$fnd_3 \longrightarrow ?$

Il s'agit de trouver W tel que :

$$(x)\ (fnd_1)\ x \wedge (x)\ (fnd_2)\ x \longrightarrow (x)\ (fnd_3 \longrightarrow W)\ x$$

$$(x)\ (fnd_1 \cdot fnd_2)\ x \longrightarrow (x)\ (\overline{fnd}_3 + W)\ x$$

$$\exists\ (fnd_3 \cdot \overline{W}) \longrightarrow \exists(\overline{fnd}_1 + \overline{fnd}_2)$$

$$fnd_3\ \overline{W} \longrightarrow \overline{fnd}_1 + \overline{fnd}_2$$

$$\overline{W} \longrightarrow \overline{fnd}_1 + \overline{fnd}_2 + \overline{fnd}_3$$

La conclusion la plus complète est obtenue en inversant la somme booléenne de tous les implicants primitifs W.

Notons $\overline{F} = \sum \overline{w}$

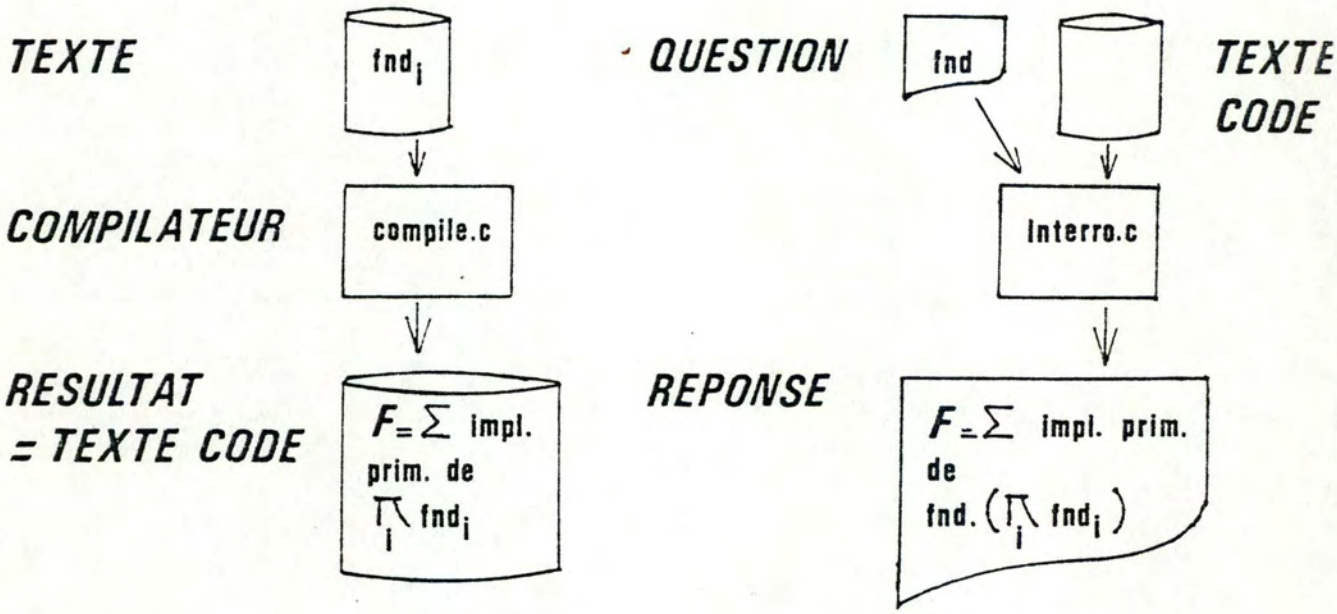
Nous pouvons alors être sûrs (voir chapitre 1) qu'en inversant $\overline{F}$ et après suppression des multiples, nous obtenons F = somme des implicants primitifs de :

$$fnd_1 \cdot fnd_2 \cdot fnd_3$$

Nous sommes assurés (voir chapitre 1) d'obtenir la conclusion F au bout des 3 étapes :

- recherche de $F_1 = \sum$ implicants primitifs de $(fnd_1 \cdot fnd_2)$
- recherche de $F_2 = \sum$ implicants primitifs de fnd_3
- suppression des multiples du produit booléen $F_1 \cdot F_2$

Le compilateur s'occupe de mener à bien la première étape.
Le programme d'exécution se charge des deux autres.



1.2. Recherche des conclusions pour prémisses existentielles

Aux prémisses existentielles de type $(\exists x)\ (X)x$, nous préférons les phrases ouvertes Xy, Xz,... qui permettent d'énumérer les éléments de l'ensemble décrit par le produit fondamental X.

Soient donc les deux prémisses :

$(x)\ (fnd)x$

Zy

Il s'agit de trouver W tel que :

$(y)\ (\ (x)\ fndx \wedge Zy \rightarrow Wy)$

$(y)\ (\exists\ \overline{fnd} \vee \bar{Z}y \vee Wy)$

$\exists \overline{fnd} \vee \bar{W} \dashv \exists Z\bar{W}$

$\bar{W} \rightarrow \bar{Z} + \overline{fnd}$

Suivant le raisonnement du paragraphe 1.1., la conclusion est :

$F = \sum$ implicants primitifs de $(Z \cdot fnd)$

1.3. Simplifications apportées par le calcul des implicants primitifs

La recherche des implicants primitifs se fait par l'algorithme décrit au chapitre 1 :

- effectuer une suppression de multiple sur la suite S_0 des monômes
- I = 1
- Calculer S_I; effectuer une suppression de multiples sur S_I
- tant que $S_I \neq S_{I-1}$
  - I = I + 1
  - Calculer SI
  - Effectuer une suppression de multiples sur SI

Cet algorithme produit automatiquement les simplifications :

$X + X = X$

$X + XY = X$

$X + \bar{X}Y = X + Y$

où X, Y sont des produits fondamentaux.

2. Définition du langage de programmation

Il s'agit de définir un langage permettant d'écrire les prémisses sous une forme aisément reconnaissable par l'ordinateur (ce qui exclut toute grammaire naturelle dont la richesse obligerait à distinguer participe passé, attribut, complément d'objet direct, etc...).

Le langage adopté est suffisamment explicite que pour être facilement accepté de l'utilisateur.
Voici écrite dans le formalisme BNF, la définition syntaxique de ce langage.

```
<texte> :: = <prémisse> $ / <prémisse> ; <texte>
<prémisse> :: = <forme universelle> / <forme existentielle>
<forme universelle> ::= > sp <forme implication> /> sp <forme
                          pourtoutx>
<forme implication> ::= <forme normale disjonctive> → sp <forme
                        normale disjonctive>
<forme pourtoutx> :: = (x) sp <forme normale disjonctive>
<forme normale disjonctive> :: = <conjonction> / <conjonction> +
                                 sp <forme normale disjonctive>
<conjonction> :: = <mot> / <mot> , sp <conjonction>
<mot> :: = <terme> / <négation de terme>
<négation de terme> :: = - <terme>
<terme> :: = <caractère admis> / <caractère admis> <terme>
<caractère admis> :: = sp/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/G/H/
                       I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z/a/
                       b/c/d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/
                       u/v/w/x/y/z
<forme existentielle> :: = <conjonction> ( <éléments> )
<éléments> :: = <élément> / <élément> , sp <éléments>
<élément> :: = <caractère admis> / <caractère admis> <élément>
```

remarques : - l'abréviation sp (space) représente le caractère blanc

- tout texte écrit dans ce langage se termine par le symbole $. Celui-ci, placé au bon endroit, permet à l'utilisateur de ne considérer qu'un sous-ensemble des prémisses contenues dans son texte.

- plutôt que d'autoriser des formes telles que (∃x) (mortel, homme) (x), nous préférons énumérer les éléments x.
  Ainsi : mortel, homme (Socrate, Aristote)
  signifie que Socrate et Aristote sont mortels et sont des hommes.
- les formes (F + G) (<éléments>) ne sont pas directement admises par le langage. Nous verrons dans le paragraphe consacré au langage d'interrogation comment remédier à cette difficulté.

2.1. Algorithmes d'analyse du langage

Voici la même grammaire écrite sous forme de règle de production

T → P $
T → P ; T

P → FU
P → FE

FU →> FI
FU →> FP

FI → FND → FND
FP → (x) FND

FND → CO
FND → CO + FND

CO → M
CO → M, CO

M → TE
M → TE

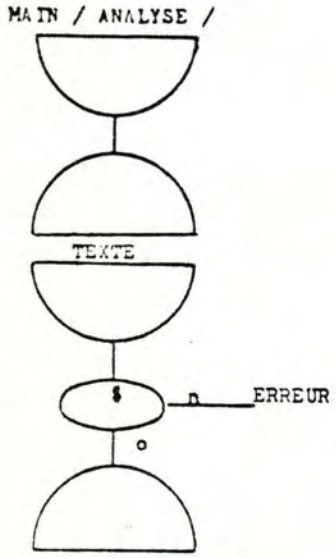
TE → CA
TE → CA TE

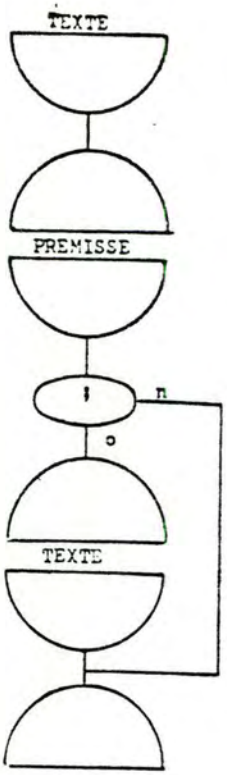
FE → CO (ELTS)

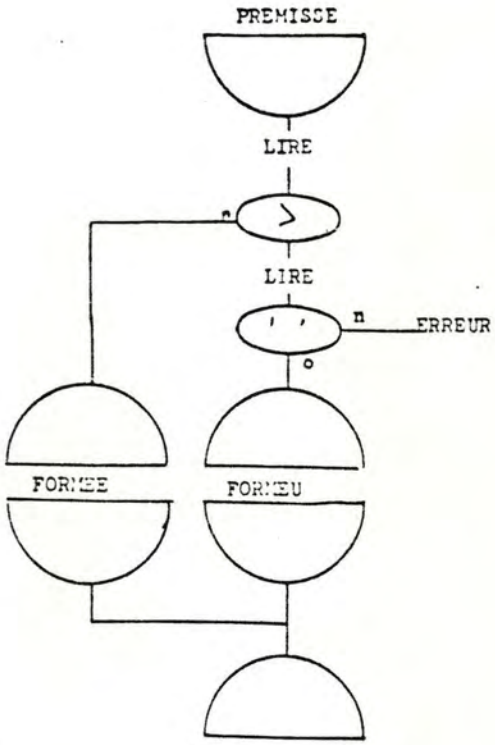
ELTS → ELT
ELTS → ELT, ELTS

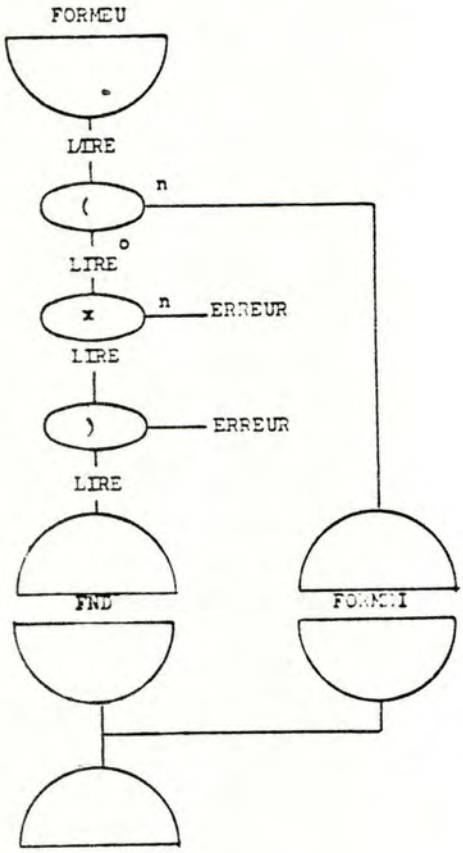
ELT → CA
ELT → CA ELT

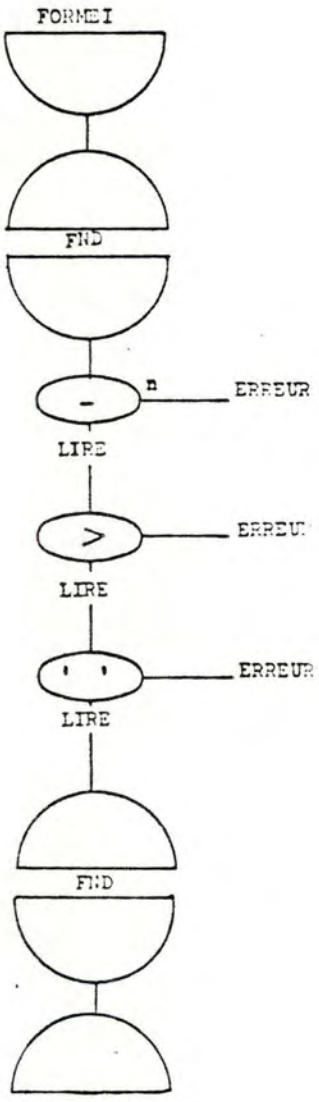
L'algorithme d'analyse du langage se fait en construisant une procédure pour chaque symbole non terminal.

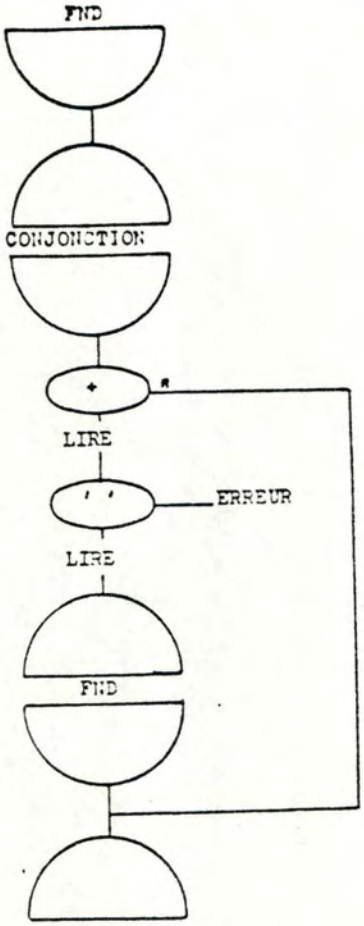

MAIN / ANALYSE /
TEXTE
$
n
ERREUR
o

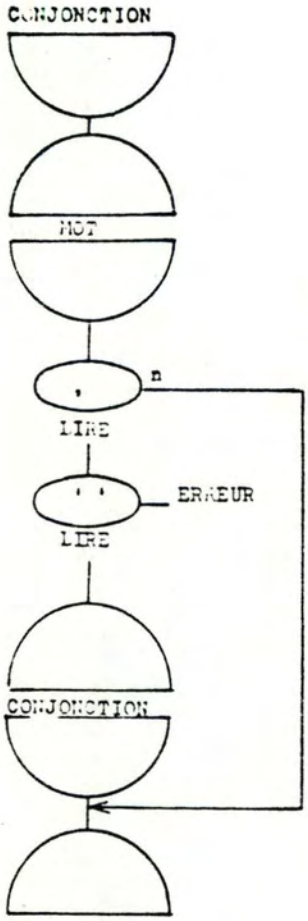

TEXTE
PREMISSE
!
n
o
TEXTE

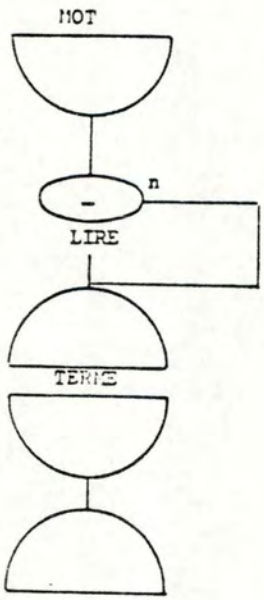

PREMISSE
LIRE
>
n
LIRE
' '
n
ERREUR
o
FORMEE
FORMEU

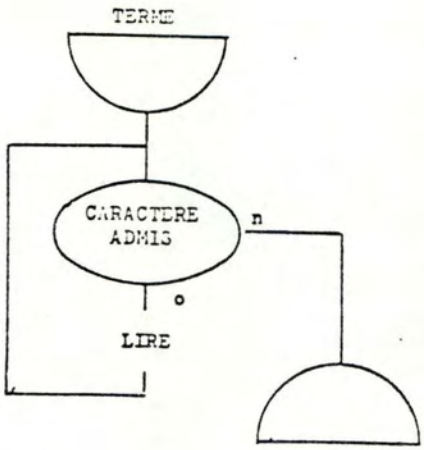

FORMEU
LIRE
(
n
o
LIRE
x
n
ERREUR
LIRE
)
ERREUR
LIRE
FND
FORMEI

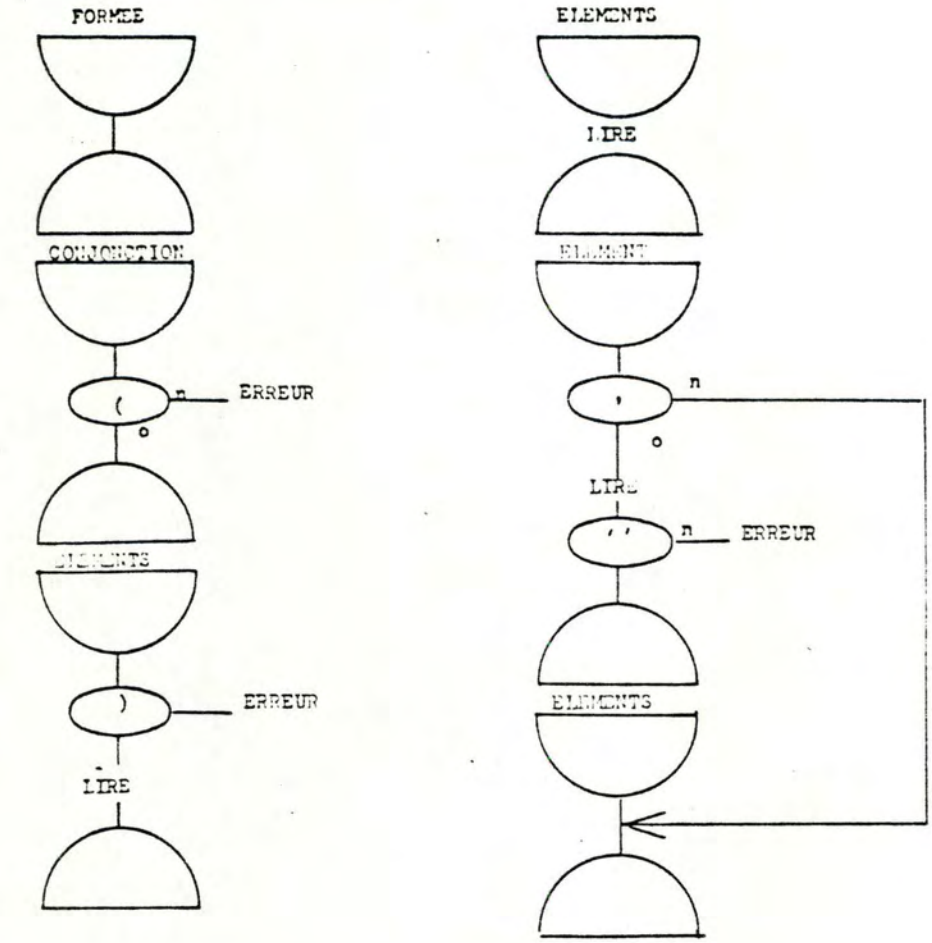

FORMEI
FND
-
n
ERREUR
LIRE
>
ERREUR
LIRE
' '
ERREUR
LIRE
FND

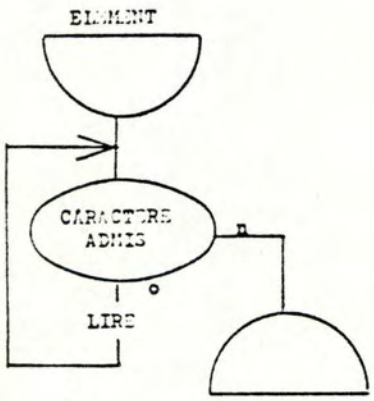

FND
CONJONCTION
+
n
LIRE
' '
ERREUR
LIRE
FND


CONJONCTION
NOT
,
n
LIRE
' '
ERREUR
LIRE
CONJONCTION

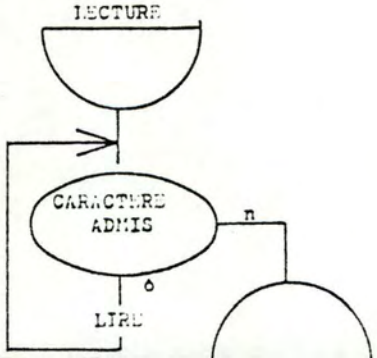

NOT
-
n
LIRE
TERME


TERME
CARACTERE
ADMIS
n
o
LIRE





Nous pouvons réécrire les procédures TERME et ELEMENT sous le nom unique LECTURE



3. Le compilateur

3.1. Codage des prémisses universelles

Les prémisses universelles "(x) (fnd) x" sont représentées dans un tableau de vecteurs (de caractères).
Chaque vecteur représente un monôme de la forme normale fnd.
Au sein d'un vecteur, toutes les variables booléennes sont représentées par un doublet de bits :

variable affirmée	(x)	0 1
variable niée	(x')	1 0
valeur nulle	(xx')	0 0
valeur indifférente	(x + x')	1 1

Les termes sont mémorisés dans le tableau "transfterme", dans l'ordre de leur apparition dans le texte.
La position d'une variable booléenne (d'1 doublet de bits) dans un vecteur codé fournit l'indice sous lequel a été enregistré le terme représenté par la variable.
exemple : soit à coder le produit des prémisses :

homme → mammifère;
mammifère → animal;
animal → être vivant.

Il s'agit de coder la forme normale :

- homme, - mammifère, - animal (1)
+ - homme, - mammifère, être vivant (2)
+ - homme, animal, être vivant (3)
+ mammifère, animal, être vivant (4)

dans les tableaux:

code FU

	1	2	3	4	
H	1 0	1 0	1 0	1 1	1 1
M	1 0	1 0	1 1	0 1	1 1
A	1 0	1 1	0 1	0 1	1 1
	1 1	0 1	0 1	0 1	1 1

transfterme

homme
mammifère
animal
être vivant

(le produit des prémisses universelles est codé dans le tableau code FU).

propriétés de cette représentation

1. Le produit booléen, bit à bit, de deux vecteurs codés contient 1 paire (00) si et seulement si le produit des deux monômes correspondants est nul.

   le produit bit à bit du premier et du dernier vecteur de l'exemple précédent donne :

1	0	1	0	1	0	1	1
1	1	0	1	0	1	0	1

1	0	0	0	0	0	1	1	**valeur nulle**

2. La somme booléenne, bit à bit, de deux vecteurs codés fournit les éléments littéraux communs aux deux monômes correspondants. Cette opération correspond donc à une "mise en évidence".

   la somme bit à bit du deuxième et du dernier vecteur de l'exemple précédent donne :

	1	2	3	4	↓		
.homme,.mamifère,être vivant (2)	1	0	1	0	1	1	0	1	1	1
mammifère,animal,être vivant (4)	1	1	0	1	0	1	0	1	1	1

être vivant	1	1	1	1	1	1	0	1	1	1

3. Le reste d'une mise en évidence est obtenu en sommant les vecteurs au vecteur inverse du vecteur-résultat de la mise en évidence.

   exemple :

	1	0	1	0	1	1	0	1	1	1
	1	1	0	1	0	1	0	1	1	1

	0	0	0	0	0	1	0	0	0

.homme,.mammifère	1	0	1	0	1	1	1	1	1
mammifère,animal	1	1	0	1	0	1	1	1	1

4. Un vecteur A est multiple d'un vecteur B si et seulement si le produit booléen bit à bit des 2 vecteurs est égal au vecteur A.
   exemple :

0 1	1 0	1 1
0 1	1 0	1 0
0 1	1 0	1 0

5. Le produit de deux vecteurs codés procure automatiquement les simplifications : $1.X = X$
   $0.X = 0$
   $X.X = X$
   $x\ x' = 0$
   où X est un produit fondamental et x une variable booléenne

3.2. Codage des prémisses existentielles

Toute prémisse existentielle "conjonction (x,y,..., z)" est codée dans un tableau de vecteurs (de caractères).
A chaque vecteur codé correspond un seul élément x,y, ...ou z.
Les noms des éléments x,y, ...z sont mémorisés dans le tableau "transfelt", dans l'ordre de leur apparition dans le texte.
La place dans le tableau code FE d'un vecteur codé fournit l'indice sous lequel est enregistré le nom de l'élément vérifiant ce code.
Si deux ou plusieurs prémisses existentielles porte sur un même élément, le code vérifié par cet élément est le produit des codes fournis par chacune des prémisses.
exemple : soit à coder les prémisses :
mammifère (Socrate, Aristote);
mortel (Aristote).
Elles sont codées dans :

code *FE*

1	1	0	1	1	1	1	1
1	1	0	1	1	1	1	0	1

transfterme

homme
mammifère
animal
être vivant
mortel

transfelt

socrate
aristote

Il reste à multiplier le vecteur code de chaque élément avec le code des prémisses universelles (code FU). Une mise en évidence nous donne alors les conclusions à mémoriser dans code FE.
exemple :

produit (code*FU* , code*FE* (0))

1	0	0	0	1	0	1	1	1
1	0	0	0	1	1	0	1	1
1	0	0	1	0	1	0	1	1
1	1	0	1	0	1	0	1	1

produit (code*FU* , code*FE* (1))

1	0	0	0	1	0	1	1	0	1
1	0	0	0	1	1	0	1	0	1
1	0	0	1	0	1	0	1	0	1
1	1	0	1	0	1	0	1	0	1

simplification

1	0	0	1	0	1	0	1	1
1	1	0	1	0	1	0	1	1

1	0	0	1	0	1	0	1	0	1
1	1	0	1	0	1	0	1	0	1

mise en évidence

1	1	0	1	0	1	0	1	1

mammifère, animal, être vivant

1	1	0	1	0	1	0	1	0	1

mammifère, animal, être vivant, mortel

si le vecteur codé d'un élément contient 1 paire (00), c'est que cet élément ne peut exister et son vecteur codé est remplacé par 1 chaîne de bits 1.

3.3. Contrôle sémantique

Après analyse et codage du texte, il est possible de savoir si certaines prémisses se contredisent.

1. Si deux ou plusieurs prémisses universelles sont logiquement contradictoires, alors leur produit donne un vecteur codé contenant au moins une paire (00)
   exemple : $(A \rightarrow B) \wedge (x)\ \bar{B}A$
   $(\bar{A} + B)\ \bar{B}A$ valeur nulle

2. Si deux ou plusieurs prémisses existentielles sont logiquement contradictoires, alors le produit des codes qu'elles génèrent contient au moins une paire (00)
   exemple : $A\ y \wedge \bar{A}\ y$
   $A\bar{A}\ y$ valeur nulle
   y ne peut exister

3. Si une prémisse existentielle est en contradiction avec une prémisse universelle, alors le produit de son vecteur codé avec code FU donne un vecteur contenant au moins une paire (00)
   exemple : $(A \rightarrow B) \wedge A\bar{B}y$
   $(\bar{A} + B)\ A\bar{B}y$
   $(\bar{A}A\bar{B} + A\bar{B}\bar{B})\ y$ valeur nulle
   y ne peut exister

Deux contrôles sémantiques sont garantis par le calcul des implicants primitifs :

4. Si une prémisse universelle est une tautologie, la recherche des implicants primitifs l'élimine.
   exemple : $(A \rightarrow B) \wedge (A \rightarrow A)$
   $(\bar{A} + B)\ (\bar{A} + A)$
   $\bar{A} + B\bar{A} + BA$
   implicants pr. : $\bar{A}, B$

5. Si toutes les prémisses universelles sont des tautologies, la recherche des consensus fait tôt ou tard apparaître un monôme de valeur = 1.

exemple : $(AB \Rightarrow AB)\ (B \Rightarrow B)$
$(\bar{A} + \bar{B} + AB)\ (\bar{B} + B)$
$(\bar{A}\bar{B} + \bar{A}B + \bar{B} + AB)$

suppression m : $\bar{A}B,\ \bar{B},\ AB$
consensus : $\bar{A}B,\ \bar{B},\ AB,\ \bar{A},\ B,\ A$
suppression m : $\bar{B},\ B,\ \bar{A},\ A$
consensus : $\underbrace{\quad}_{1}$

3.4. La compilation

La constante MAXSIZE définit le nombre maximum de caractères utilisables pour coder un monôme.
Un caractère contenant 8 bits, le nombre maximum de termes mémorisables est
4 * MAXSIZE.

La constante placeréservée = 50 * MAXSIZE.

Définition des variables globales
++++++++++++++++++++++++++++++++

zone travail : zone de travail utilisée dans les différentes procédures (longueur = placeréservée + 1).

code FE : place réservée pour contenir le code des prémisses existentielles.
code FE [i] = chaîne de (MAXSIZE + 1) caractères, contient le code du $i^{è}$ élément (le dernier caractère étant '\0').
MAXSIZE * 24 éléments sont mémorisables, au maximum.

code FU : place réservée pour contenir le code du produit des prémisses universelles (longueur = placeréservée + 1).

pourtoutx : place réservée pour contenir le code de la prémisse universelle courante (longueur = placeréservée + 1).

transfterme : place réservée pour enregistrer le nom des termes contenus dans le texte.
4 * MAXSIZE termes sont mémorisables au maximum.
transfterme [i] = chaîne de caractères contenant le nom du $i^{è}$ terme.
Seuls les 9 premiers caractères d'un terme sont mémorisés. (le dernier caractère de la chaîne transferterme [i] étant obligatoirement le caractère spécial '\0').

transfelt : place réservée pour enregistrer les éléments contenus dans le texte. 24 * MAXSIZE éléments sont mémorisables au maximum.
transfelt [i] = chaîne de caractères contenant le $i^{è}$ élément.
Seuls les 9 premiers caractères d'un élément sont mémorisés (le dernier caractère de la chaîne transfelt [i] étant obligatoirement le caractère spécial'\0')

erreur : cette variable est initialisée à 0; elle vaut 1 dès qu'une erreur est détectée dans le texte.

c : contient le caractère courant lu dans le fichier contenant le texte à coder.

table : est une liste de piles.

pile : est une liste d'entiers;
table et pile sont utilisés pour analyser toute forme normale disjonctive (voir procédures FND et CONJONCTION).

main ( )
++++++++

fonction : assurer la coordination entre les différentes procédures dans le but :
1. d'analyser un texte écrit dans le langage de programmation.
2. si le texte n'est pas syntaxiquement correct, avertir l'utilisateur et énumérer les fautes de syntaxe.

3. si le texte est syntaxiquement correct :
   coder les prémisses existentielles et universelles.
4. si le texte n'est pas sémantiquement correct :
   donner à l'utilisateur le type d'erreur sémantique.
5. si le texte est sémantiquement correct :
   enregistrer le code généré au point 3

argument : chaîne de caractères titrein.

precond : "titrein" est terminée par le caractère spécial ' 0'.

résultat : un des messages suivants :

1. ce texte n'existe pas
2. fin de texte introuvable
3. en-tête de la prémisse universelle [(x)]
4. en-tête de la prémisse existentielle
5. forme implication non correcte
6. fin de la prémisse existentielle
7. attention; un blanc après le symbole '+'
8. attention; un blanc après le symbole ','
9. attention; un blanc après le symbole '>'
10. trop de termes employés : MAXSIZE * 4 au max.
11. trop d'éléments employés : MAXSIZE * 24 au max.
12. pas assez de place : modifier placeréservée
13. il existe une prémisse en contradiction avec les autres
14. l'ensemble des prémisses universelles forme 1 tautologie
15. l'élément suivant ne peut exister

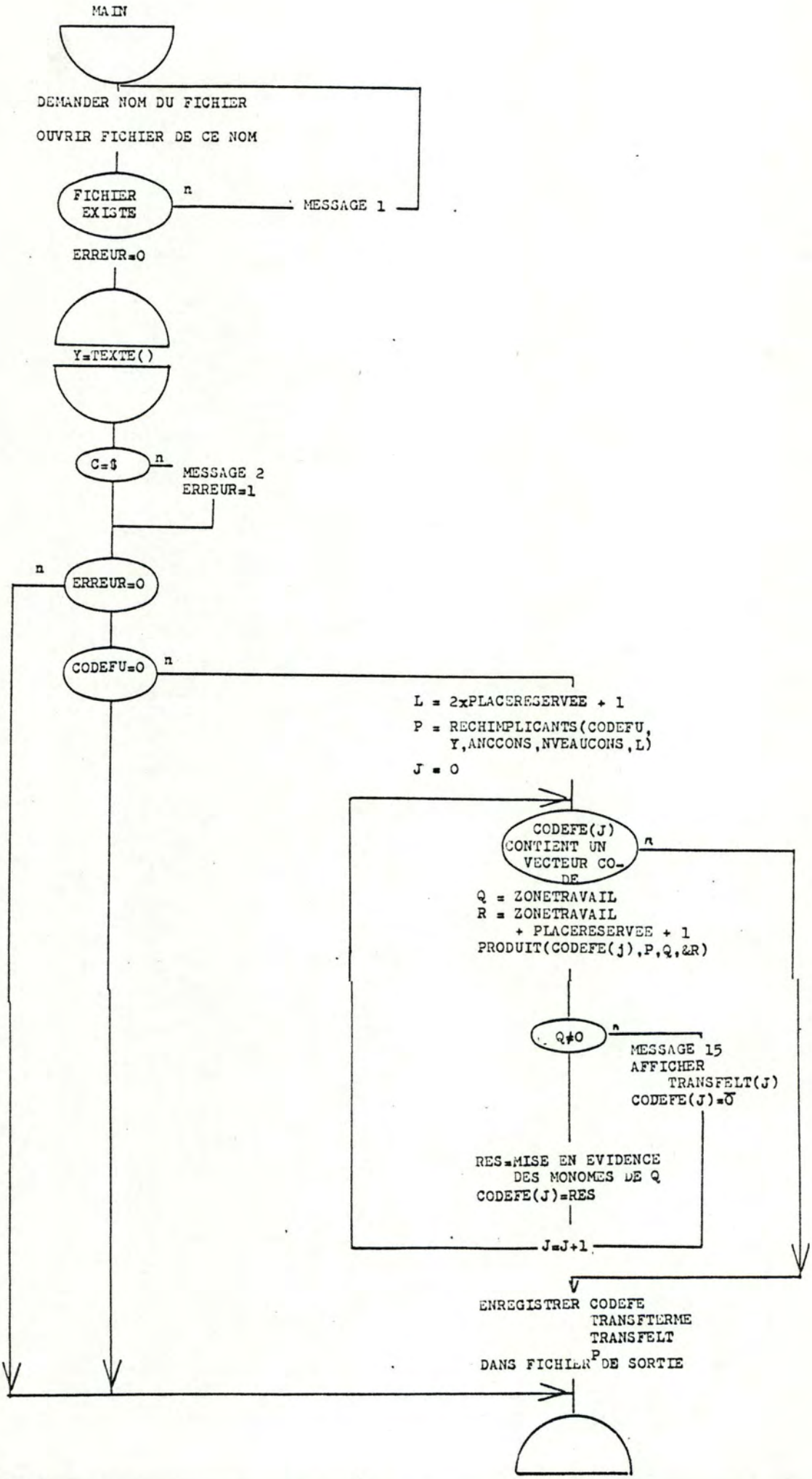
ou un fichier de sortie contenant le code des prémisses universelles et existentielles.

postcond :

message 1 si le fichier de nom "titrein" n'existe pas.
ou message 1 à 9 si le texte du fichier "titrein" est syntaxiquement non correct.
ou message 10 à 12 si la place réservée pour mémoriser le code est trop restreinte.
ou message 13 à 15 si le texte du fichier "titrein" est sémantiquement incorrect.

ou fichier de sortie :

si aucune erreur de syntaxe n'a été repérée, et si les prémisses universelles ne sont pas contradictoires;
ce fichier contient le code du texte de "titrein",
c'est-à-dire :
les termes (contenus dans transfterme),
les éléments (contenus dans transfelt),
le code de chaque élément (contenu dans code FE),
les implicants primitifs du produit des prémisses universelles (produit contenu dans code FU);
le nom du fichier est la concaténation "titrein" avec le groupe de lettre '.o' .


MAIN
DEMANDER NOM DU FICHIER
OUVRIR FICHIER DE CE NOM
FICHIER EXISTE
n
MESSAGE 1
ERREUR=0
Y=TEXTE()
C=$
n
MESSAGE 2
ERREUR=1
n
ERREUR=0
CODEFU=0
n
L = 2xPLACERESERVEE + 1
P = RECHIMPLICANTS(CODEFU, Y,ANCCONS,NVEAUCONS,L)
J = 0
CODEFE(J) CONTIENT UN VECTEUR CODE
n
Q = ZONETRAVAIL
R = ZONETRAVAIL + PLACERESERVEE + 1
PRODUIT(CODEFE(j),P,Q,&R)
Q≠0
n
MESSAGE 15
AFFICHER TRANSFELT(J)
CODEFE(J)=0
RES=MISE EN EVIDENCE DES MONOMES DE Q
CODEFE(J)=RES
J=J+1
ENREGISTRER CODEFE TRANSFTERME TRANSFELT
DANS FICHIER P DE SORTIE

rechimplicants
++++++++++++++

fonction : rechercher les implicants primitifs d'une forme normale disjonctive F.

arguments : un tableau de caractères a,
un tableau de caractères y,
un tableau de caractères acons,
un tableau de caractères ncons,
un entier n.

precond : a contient le code de F (terminé par'\0',
y contient un seul élément :'\0',
acons et ncons sont vides,
n est la longueur du tableau acons et ncons.

résultat : un tableau de caractères X ou message d'erreur.

postcond : X est un des tableaux acons ou ncons ;
X est terminé par'\0';
X contient les implicants primitifs de F si
F est différent de 1,
F est suffisamment grand pour contenir tous les implicants
si F = 1:X contient 1 seul vecteur codé constitué d'une chaîne de bits 1 + message 14
si X ne peut contenir tous les implicants : message 12.

```
P = cons + (2 * placeréservée + 1)
tant que ( (y ≠ a) et (erreur = 0) )
            consensus (a,y, cons, P)
            q = acons + n
            suppressionmltple (cons, a, acons, &q)
            a = acons
            y = q
            acons = ncons
            ncons = a
```

consensus
+++++++++

fonction : rechercher les consensus entre les monômes d'1 forme normale disjonctive $F_1 + F_2$.

arguments : 1 tableau de caractères a,
1 chaîne de caractères y,
1 tableau de caractères cons,
1 pointeur vers caractère P.

precond : a contient le code de F_1 suivi de la chaîne y;
y contient le code de F_2 (terminé par '\0');
F_2 ne contient que des implicants primitifs;
cons est vide;
P pointe vers le dernier caractère de cons.

résultat : message d'erreur
ou
cons n'est plus vide.

postcond : cons contient les consensus des monômes de a entre eux et les consensus des monômes de y avec ceux de a.

si cons ne peut contenir tous les consensus : erreur 12

si un consensus a la valeur 1 : erreur 14,
cons contient un seul vecteur codé constitué d'une chaîne de bits 1.

cons est terminé par'\0'.

suppressionmltple
+++++++++++++++++

fonction : supprimer les multiples d'une fnd $F = F_1 + F_2$
où F_2 est déjà débarrassé de ses multiples.

arguments : 1 tableau de caractères c,
1 tableau de caractères d,
1 tableau de caractères e,
1 pointeur vers caractère f.

precond : c contient le code de F_1 (terminé par'\0');
d contient le code de F_2 (terminé par'\0');
F_2 est déjà débarrassé de ses propres multiples;
e est vide;
f pointe vers le dernier caractère de e.

résultat : message d'erreur
ou
e n'est plus vide + f est modifié.

postcond : e contient la forme $F_1 + F_2$ (terminé par' 0')
débarrassée de ses multiples;
f pointe vers le premier monôme de F_2;
erreur 12 si e ne peut contenir le résultat.

produit
+++++++

fonction : calculer le produit de deux fnd F_1 et F_2.

arguments : 1 tableau de caractères a,
1 tableau de caractères b,
1 tableau de caractères c,
1 pointeur vers caractère d.

precond : a contient le code de F_1 (terminé par'\0');
b contient le code de F_2 (terminé par'\0');
c est vide;
d pointe vers le dernier caractère de c.

résultat : message d'erreur
ou
c n'est plus vide.

postcond : c contient le produit $F_1 . F_2$ (terminé par '\0'),
amputé de ses monômes nuls.
erreur 12 si c ne contient pas assez de place pour
mémoriser le produit $F_1 . F_2$

simplification
++++++++++++++

fonction : déterminer si une chaîne de caractères contient le
doublet de bits 00.

argument : une chaîne de caractères k.

precond : k est terminé par .

résultat : un entier s.

postcond : s = 0 si k ne contient pas de doublet 00;
s = - 1 si k contient au moins un doublet 00.

TEXTE
+++++

fonction : analyser et coder le texte du fichier "titrein".

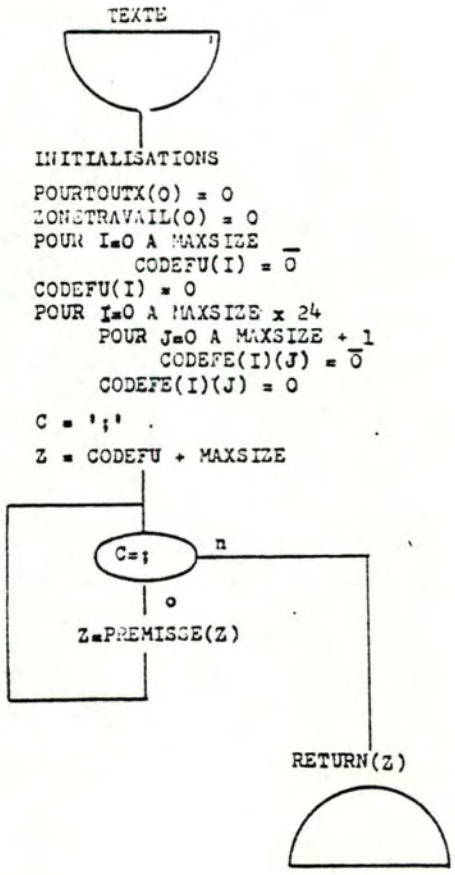
argument : fichier de nom "titrein".

precond : ce fichier existe.

résultat : message d'erreur
ou (exclusif)
code du texte + pointeur p vers caractère.

postcond : message 3 à 9 si le texte est syntaxiquement incorrect;
message 10 à 12 s'il manque de place réservée pour
mémoriser le code;
le code se trouve dans les tableaux : code FU
(terminé par'\0')
transfterme, transfelt, code FE;
pour tout i, code FE[i], transfterme[i], transfelt[i]
est terminé par '\0';
p pointe vers dernier caractère de code FU ('\0');

si les prémisses universelles sont contradictoires, alors code FU contient le seul élément '\0'; si toutes les prémisses universelles sont des tautologies, alors code FU contient un seul vecteur codé, de bits 1.



PREMISSE
++++++++

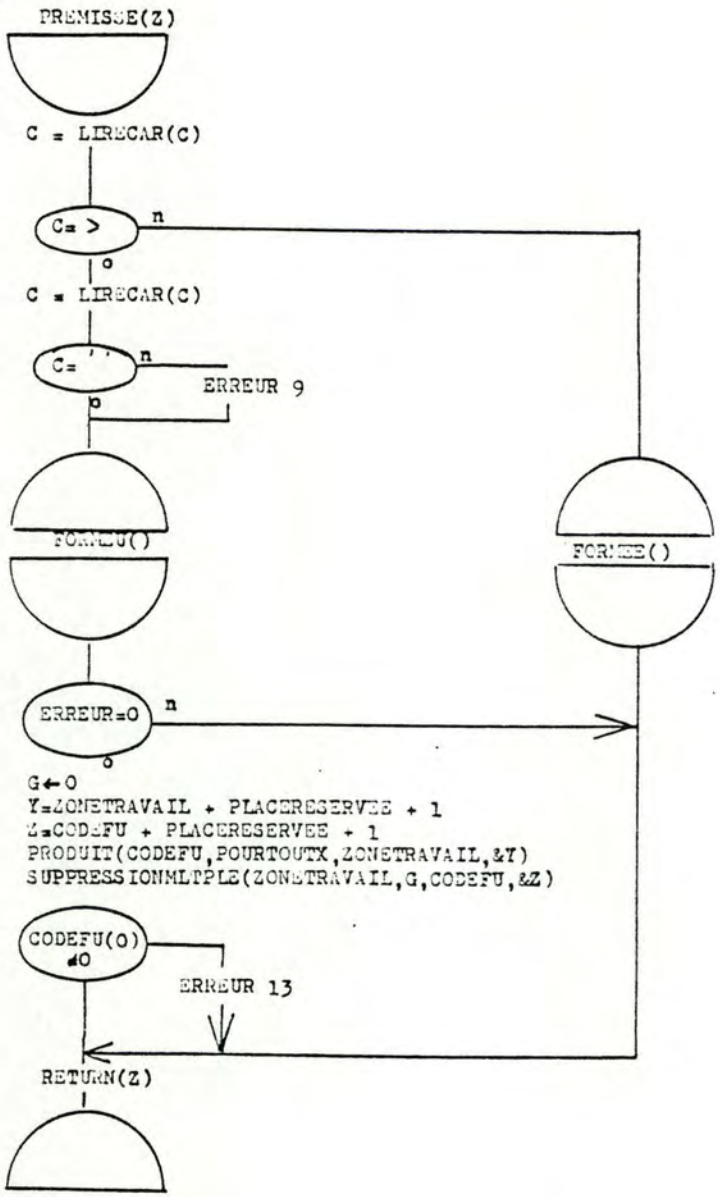
fonction : analyser et coder la prémisse courante.

argument : une prémisse du texte contenu dans fichier "titrein",
un pointeur z vers caractère.

precond : z pointe vers le dernier caractère de code FU ('\0').

résultat : message d'erreur
ou (exclusif)
code de la prémisse + pointeur z vers caractère.

postcond : message 3 à 9 si la prémisse n'est pas syntaxiquement correcte;
message 10 à 12 s'il manque de place réservée pour mémoriser le code;
message 13 si la prémisse est une prémisse universelle en contradiction avec les autres;
code de la prémisse courante : dans transfterme
dans pourtoutx si prémisse universelle
dans code FE et transfelt si prémisse existentielle;
si la prémisse courante est de type universelle :
code FU contient le produit de code FU avec pourtoutx;
tous les codes sont terminés par'\0';
z pointe vers le dernier caractère de code FU ('\0').



FORMEU
++++++

fonction : analyser et coder une prémisse universelle.

argument : une prémisse du texte contenu dans "titrein".

precond : : il s'agit d'une prémisse universelle (x) ou implication

résultat : message erreur
ou (exclusif)
code de la prémisse + mémorisation des nouveaux termes

postcond : prémisse codée dans pourtoutx (terminé par '\0') ;
nouveaux termes mémorisés dans transfterme ;
message 3, 5, 7, 8 ou 9 si prémisse syntaxiquement incorrecte ;
message 10 ou 12 s'il manque de place réservée pour mémoriser le code.

FORMEI
++++++

fonction : analyser et coder une implication.
argument : une prémisse du texte contenu dans "titrein".
precond : prémisse de type implication.
résultat : message d'erreur
ou (exclusif)
code de la prémisse + mémorisation des nouveaux termes.

postcond : prémisse codée dans pourtoutx (terminé par '\0');
termes mémorisés dans transfterme;
message 5, 7, 8 ou 9 si prémisse syntaxiquement non correcte;
message 10 ou 12 s'il manque de place réservée pour mémoriser le code.

FND
+++

fonction : analyser une forme normale disjonctive lue dans "titrein".

argument : une fnd lue dans "titrein".
precond : -
résultat : message d'erreur
ou (exclusif)
liste dont chaque élément est une liste d'entiers
+ mémorisation des nouveaux termes de fnd

postcond : message 7 ou 8 si fnd n'est pas syntaxiquement correcte
message 10 s'il manque de la place pour mémoriser les nouveaux termes
(les termes de la fnd sont mémorisés dans transfterme)
chaque liste d'entiers représente un monôme de fnd
propriétés : décrites dans les postcond de CONJONCTION.

CONJONCTION
++++++++++

fonction : analyser une conjonction de termes.
argument : une conjonction lue dans "titrein".
precond : -

résultat : message d'erreur
ou (exclusif)
liste d'entiers i + mémorisation des nouveaux termes de la conjonction.

postcond : les nouveaux termes sont mémorisés dans transfterme ;
chaque nombre i représente un terme de la conjonction ;
i = n° du terme (attribué par transf),
i > 0 si le terme est affirmé dans la conjonction,
i < 0 si le terme est nié dans la conjonction,
erreur 8 si conjonction n'est pas syntaxiquement correcte ;
erreur 10 s'il manque de place pour mémoriser les nouveaux termes.

MOT

fonction : analyser et mémoriser un terme.
argument : un terme lu dans le texte de "titrein".
precond : -
résultat : message d'erreur
ou (exclusif)
un entier i et mémorisation du terme.

postcond : message 10 s'il manque de place pour mémoriser le terme.
i = n° du terme (attribué par transfd),
i > 0 si le terme est affirmé,
i < 0 si le terme est nié ;
le terme est mémorisé dans transfterme s'il s'agit d'un nouveau terme.

LECTURE

fonction : lire une chaîne de caractères dans le fichier "titrein".
arguments : 1 chaîne de caractères d .
1 chaîne de caractères e du fichier "titrein".

precond : d est vide .
résultat : d est remplie des caractères de e.

postcond : sont mis dans d les 9 premiers caractères de e;
le dernier caractère de d est '\0';
la chaîne e est lue jusqu'au premier caractère non admis (celui-ci n'est pas enregistré dans d);
les caractères admis sont : sp
a → z
A → Z
0 → 9.

transf
++++++

fonction : attribuer un n° à un terme.
argument : une chaîne de caractère d, représentant un terme.
precond : d est terminé par '\0'
résultat : un entier i et mémorisation du terme.
postcond : si le terme a déjà été mémorisé:
i = indice de transfterme sous lequel le terme a été enregistré + 1 (i est donc toujours différent de 0)
si le terme n'a pas encore été mémorisé et s'il est possible de le faire
k = indice du dernier terme enregistré + 1,
le nouveau terme est mémorisé dans transfterme sous l'indice k,
i = k + 1
si le terme n'a pas encore été mémorisé et s'il n'y a plus de place pour le faire:
i = -1.

comparaison
++++++++++

fonction : comparer deux chaînes de caractères.
arguments : 2 chaînes de caractères c et d
precond : c et d sont terminés par '\0'.
résultat : un entier i.
postcond : i = -1 si les deux chaînes sont différentes;
i = 0 si les deux chaînes sont identiques.

coderp
++++++

fonction : coder une forme normale disjonctive F,

arguments : tableau de caractères f,
un pointeur g vers caractère,
F est représentée par une liste dont chaque élément est une liste d'entiers (fournie par FND).

precond : g pointe vers le dernier caractère de f; f est vide.

résultat : message d'erreur
ou (exclusif)
F est codée.

postcond : message 12 s'il n'y a pas assez de place pour mémoriser le code;
F est codée dans le tableau f.

codern
++++++

fonction : coder la négation $\bar{F}$ d'une forme normale disjonctive F.

arguments : tableau de caractère f,
un pointeur g vers caractère,
F est représentée par une liste dont chaque élément est une liste d'entiers (fournie par FND).

precond : g pointe vers le dernier caractère de f; f est vide.

résultat : message d'erreur
ou (exclusif)
F est codée + g est modifié.

postcond : message 12 s'il n'y a pas assez de place pour mémoriser le code;
$\bar{F}$ est codée dans le tableau f;
g pointe vers le dernier élément de f ('\0').

FORMEE
+++++++

fonction : analyser et coder une prémisse existentielle.

argument : une prémisse du texte contenu dans le fichier "titrein".

precond : il s'agit d'une prémisse existentielle.

résultat : message d'erreur
ou (exclusif)
code de la prémisse + mémorisation des nouveaux termes + mémorisation des nouveaux éléments.

postcond : message 4, 6 ou 8si la prémisse n'est pas syntaxiquement correcte;
message 10 ou 11 s'il n'y a pas assez de place pour mémoriser les termes et éléments;
les nouveaux termes sont mémorisés dans transfterme;
les nouveaux éléments sont mémorisés dans transfelt
(le vecteur code de la prémisse x code FE[i]) est codé dans code FE [i] (terminé par '\0') pour tout i = n° des éléments concernés par la prémisse.

ELEMENTS
++++++++

fonction : analyser et mémoriser une suite d'éléments.

arguments : la suite d'éléments d'une prémisse existentielle
un vecteur code T d'une prémisse existentielle.

precond : -

résultat : message d'erreur
ou (exclusif)
mémorisation des nouveaux éléments + mémorisation de (vecteur code T x code FE[i]) dans code FE[i] (terminé par '\0') pour tout i = n° des éléments

postcond : message 8 si la suite d'éléments n'est pas syntaxiquement correcte.
message 11 s'il n'y a pas assez de place pour mémoriser les éléments.

eltttransf
++++++++

fonction : attribuer un n° à un élément.

argument : une chaîne de caractère d, contenant un élément.

precond : d est terminé par '\0'

résultat : un entier i et mémorisation de l'élément.

postcond : si l'élément a déjà été mémorisé :
i = indice de transfelt sous lecuel l'élément a été mémorisé ;
si l'élément n'a pas encore été mémorisé et s'il est possible de le faire :
i = indice du dernier élément enregistré + 1
le nouvel élément est mémorisé dant transfelt sous l'indice i;
si l'élément n'a pas encore été mémorisé et s'il n'y a plus de place pour le faire :
i = -1.

lirecar
+++++++

fonction : lire caractère suivant du fichier contenant le texte à analyser et coder.

argument : c = dernier caractère lu.

precond : -

résultat : caractère d.

postcond : si c = EOF; d=c
si c ≠ EOF; c = caractère suivant dans le fichier si ce caractère est différent de <tab> et différent de <return> .

4. Définition du langage d'interrogation

Le langage doit être capable :

(1) de demander la liste des éléments enregistrés, appartenant à un ou plusieurs termes.
(par exemple, demander la liste des hommes, mortels).

(2) de demander les termes auquel appartient un élément quelconque enregistré.
(par exemple, demander ce qu'est Socrate).

(3) de demander, étant donné une forme normale disjonctive de termes, les conclusions qu'elle implique.

A chacune de ces questions correspond une des formes :

(1) > <conjonction> ?
(2) ? <élément>.
(3) <forme normale disjonctive> →

Les deux premières questions ne nécessitent aucun traitement particulier : leurs réponses ont été codées et enregistrées par le compilateur.

Attardons-nous à la troisième forme

- L'absence de phrase<forme normale disjonctive> (<éléments>) dans le langage de programmation n'est pas trop contraignante.
  En effet, si nous désirons des conclusions sur un élément vérifiant une fnd F, il suffit de poser la question 'F→'

- La réponse à ce type de question est une forme normale disjonctive dans laquelle les termes communs sont mis en évidence.

- La réponse à la question 'fnd→' est obtenue en exécutant les étapes :
  coder fnd→ f
  rechercher les implicants primitifs de f→ F
  multiplier F par G→ H
  opérer une suppression de multiples dans H→ I
  opérer une mise en évidence dans I
  G est le code des implicants primitifs du produit des

prémisses universelles, enregistré lors de la compilation.

exemple : - soient les prémisses

Dieu → -mortel (D → -M)

mortel → vivant (M → V)

homme → mortel (H → M)

animal → mortel (A → M)

- le code enregistré par le compilateur est :
  $\bar{M}\bar{H}\bar{A} + \bar{D}\bar{H}\bar{A}V + \bar{D}\bar{M}V + \bar{H}\bar{A}\bar{D}V$
- la réponse à la question A+ V→ est :
  $V\ (\bar{M}\bar{H}\bar{A} + \bar{D}\bar{H}\bar{A} + \bar{D}\bar{M} + \bar{H}\bar{A}\bar{D})$
  Si à la suite de cela, la question 'D→' est posée, la réponse est $DV\bar{M}\bar{H}\bar{A}$
- la réponse à la question 'AD→' est un monôme nul, ce qui signifie qu'il n'existe aucun élément qui soit AD.

Les questions de ce type peuvent s'emboîter. C'est pourquoi nous ajoutons les questions :

(4) # Il s'agit d'une demande d'affichage du contexte courant. Le contexte est représenté par la suite des antécédents des questions (de type (3)) posées

(5) < Il s'agit d'une demande de retour au dernier contexte, précédant le contexte courant. Elle ne permet pas de remonter jusqu'à l'avant dernier contexte; elle peut servir à éliminer la dernière question de type (3).

(6) : Il s'agit d'une demande de retour au contexte initial.

(7) $ Il s'agit d'une demande d'arrêt du programme.

Définition syntaxique

<question> → <demande d'élts>

<question> → <demande de termes>

<question> → <demande de conclusion>

<question> → <retour au contexte initial>

<question> → <retour au contexte précédent>

<question> → <demande d'arrêt>

<question> → <demande affichage contexte>

<demande d'élts> → > sp <conjonction> ?

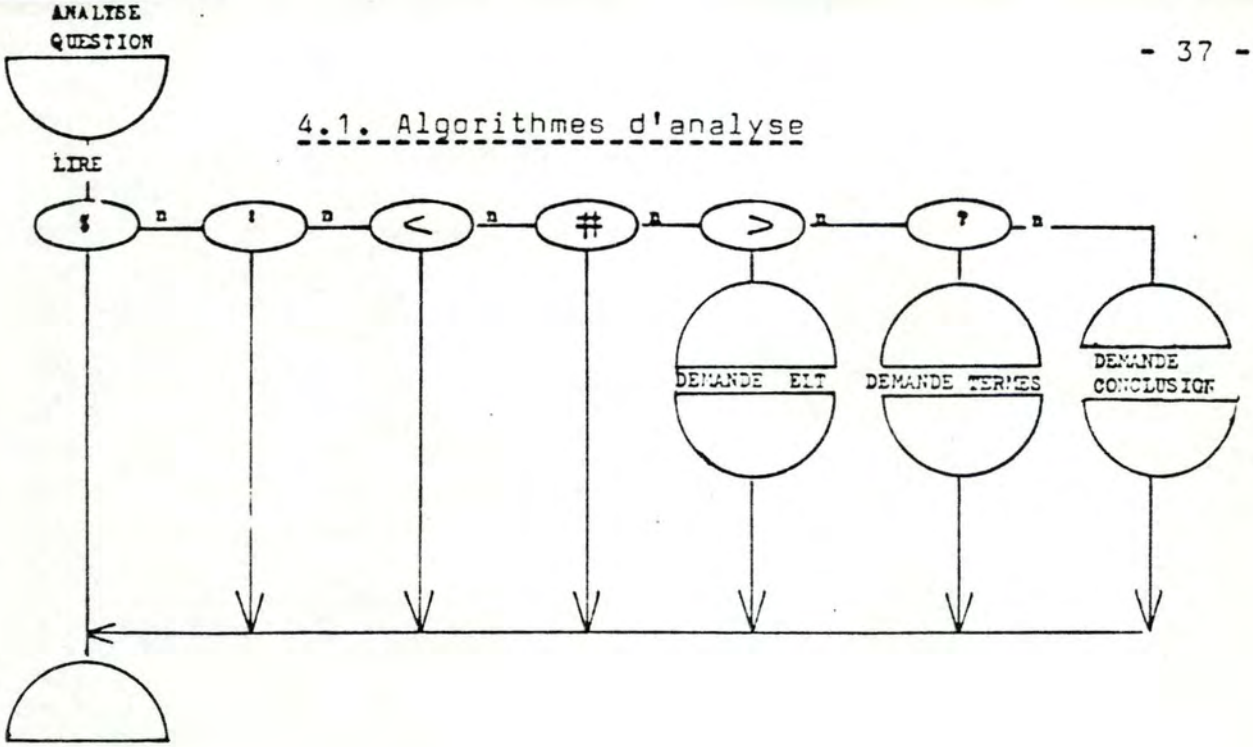
<demande de termes> → ? sp <élément>.

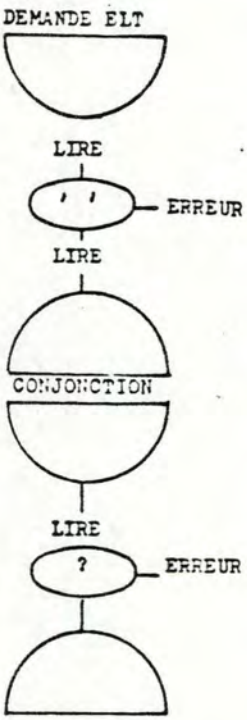
<demande de conclusion> → <fnd> →

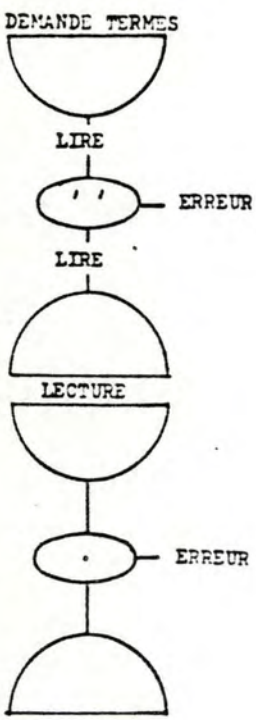
<retour au contexte initial> → :
<retour au contexte précédent> → <
<demande d'arrêt> → $
<demande affichage contexte> → #
<fnd> → <conjonction>
<fnd> → <conjonction> + sp <fnd>
<conjonction> → <mot>
<conjonction> → <mot> , sp <conjonction>
<mot> → <terme>
<mot> → - <terme>
<terme> → <caractère admis>
<terme> → <caractère admis> <terme>
<élément> → <caractère admis>
<élément> → <caractère admis> <élément>
<caractère admis> → sp/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/G/H/I/
J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z/a/b/c/
d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/
x/y/z

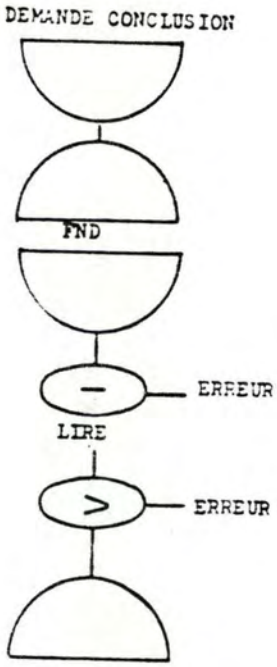
4.1. Algorithmes d'analyse

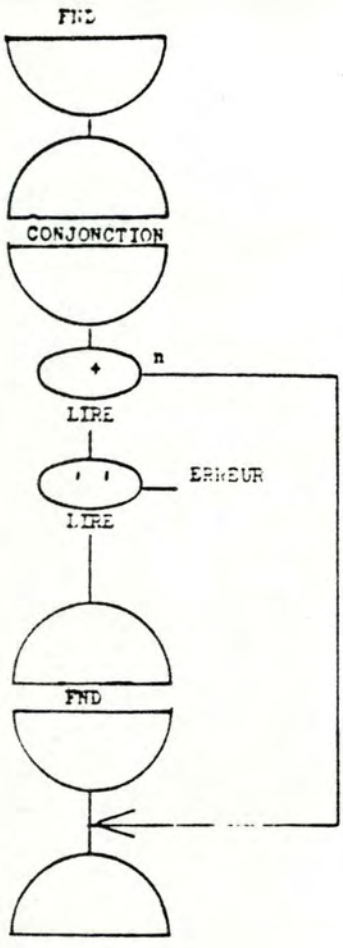
4.1. Algorithmes d'analyse

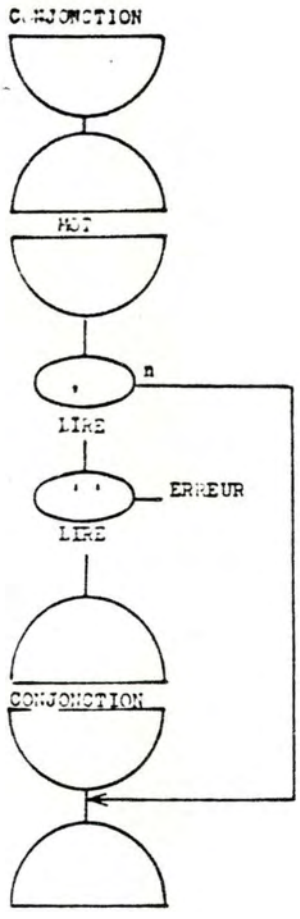


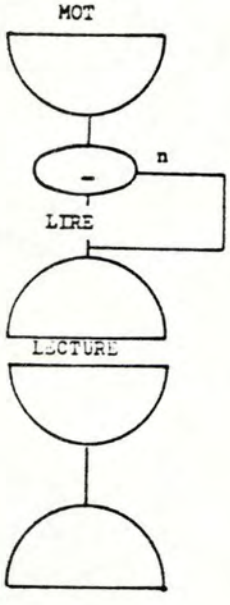


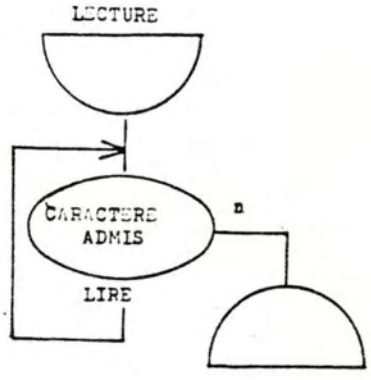





FND
CONJONCTION
+
n
LIRE
' '
ERREUR
LIRE
FND


CONJONCTION
MOT
,
n
LIRE
' '
ERREUR
LIRE
CONJONCTION


MOT
-
n
LIRE
LECTURE


LECTURE
CARACTERE ADMIS
n
LIRE

5. Le programme d'exécution

la constante MAXFORMAT définit le nombre maximum de caractères utilisables pour coder un monôme.

la constante placeréservée = 50 * MAXFORMAT.

Définition des variables globales

code FE contient le code des prémisses existentielles.
code FE[i] (= chaîne de MAXFORMAT + 1 caractères)contient le code du $i^{è}$ élément (le dernier caractère étant'\0).
Il y a au maximum 24 * MAXFORMAT éléments.

transfterme transfterme [i] = chaîne de caractères contenant le $i^{è}$ terme (le dernier caractère étant'\0').
Il y a au maximum 4 * MAXFORMAT termes.

transfelt transfelt [i] = chaîne de caractères contenant le $i^{è}$ élément (le dernier caractère étant'\0').
Il y a au maximum 24 * MAXFORMAT éléments.

codeinitial tableau contenant le code des prémisses universelles.
(longueur =2 * placeréservée + 1).

zone 1 et zone 2 zones de travail utilisées dans les différentes procédures.
(longueur = 2* placeréservée + 1).

codecrt pointe vers la zone de mémoire contenant le code courant des prémisses universelles.

codepct pointe vers la zone de mémoire contenant le dernier code des prémisses universelles, précédant le code courant.

contextecrt tableau contenant le contexte courant, codé de la façon suivante :
chaque antécédent d'une question de type (3) est une fnd; chaque fnd est suivi, dans le tableau, par un groupe de 2 caractères '\0';
chaque monôme d'une fnd est séparé des autres par un caractère '\0';
chaque terme, au sein d'un monôme, est représenté par l'indice i sous lequel il est mémorisé dans transfterme;
i> 0 si le terme est affirmé, i<0 s'il est nié.

ctxtcrt pointeur qui délimite le contexte courant; il pointe, dant contextecrt, vers le premier caractère qui suit le dernier groupe de 2 caractères '\0'.

ctxtpct pointeur qui délimite le dernier contexte avant le contexte courant; il pointe, dans contextecrt, vers le premier caractère qui suit l'avant dernier groupe de 2 caractères '\0'.

quesconcl place réservée pour contenir le code d'une question de type <demande de conclusion> (longueur = placeréservée+1

queselt vecteur de (MAXFORMAT + 1) caractères contenant le code d'une question de type <demande d'élts> (longueur = placeréservée + 1).

table liste de piles.

pile liste d'entiers.
table et pile sont utilisés pour analyser une forme normale disjonctive (voir procédures fnd et conjonction).

erreur cette variable est initialisée à 0; elle vaut 1 dès qu'une erreur est détectée.

c contient le caractère courant, lu sur l'écran.

main ( )
++++++++

fonction : analyser les questions de l'utilisateur dans le but d'y répondre.

argument : 1 nom de fichier
et
1 question.

précond : -

résultat : message d'erreur
ou (exclusif)
réponse à la question.

postcond : il s'agit d'un des messages suivants :
1. ce texte n'est pas codé
2. MAXSIZE est différent de MAXFORMAT : modifier ce dernier
3. il manque un signe d'implication '→ '
4. il manque un blanc après le signe '→ '

5. il manque un '?'
6. il manque un blanc après le '?'
7. il manque un blanc après '>'
8. cet élément n'existe pas
9. certains termes n'existent pas
10. il manque un blanc après '+'
11. il manque un blanc après ','
12. pas assez de place : modifier placeréservée
13. il manque un '.'

analyse
+++++++

fonction : analyser une question lue sur l'écran et y répondre.

arguments : une chaîne de caractères d, lue sur l'écran.

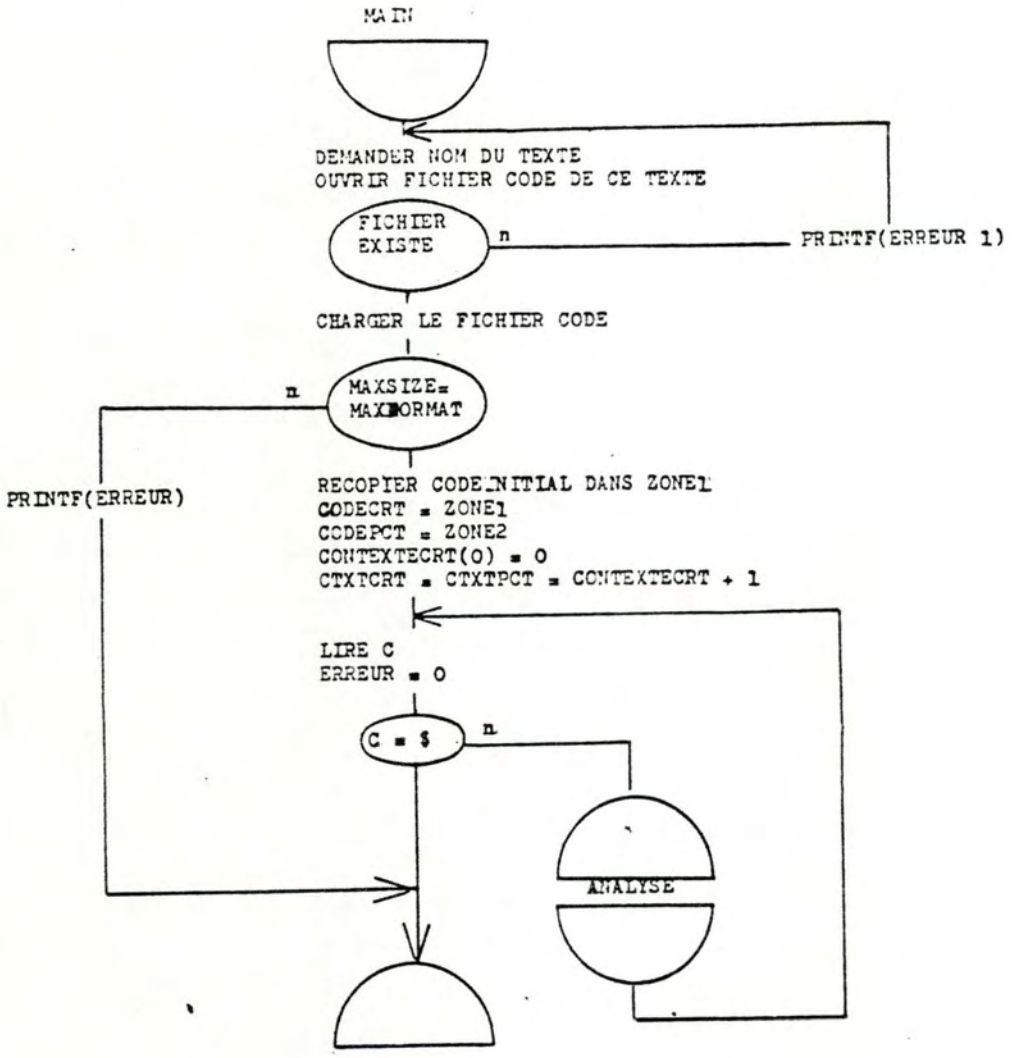
precond : -

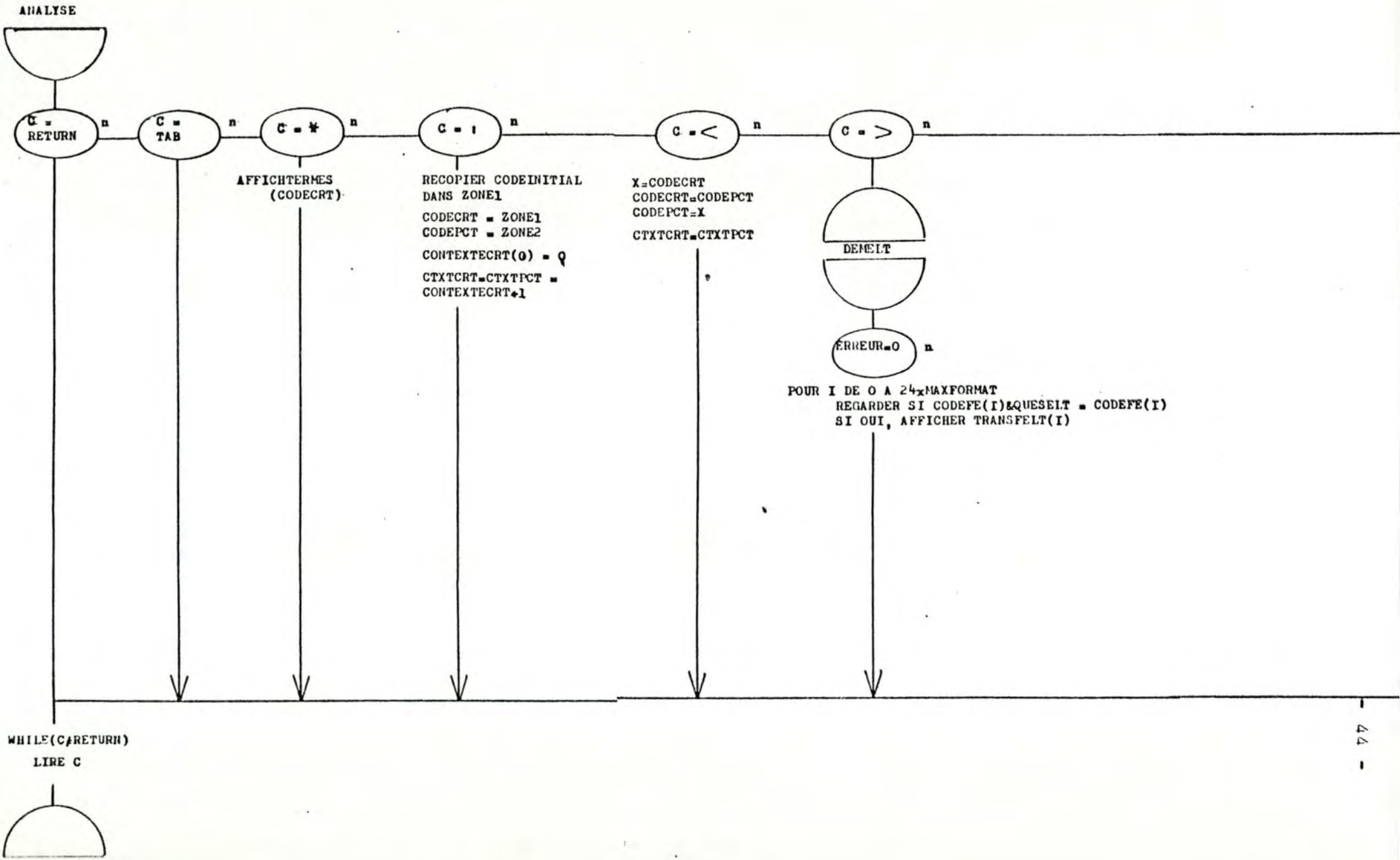
résultat : réponse à la chaîne de caractère d
ou (exclusif)
message d'erreur.

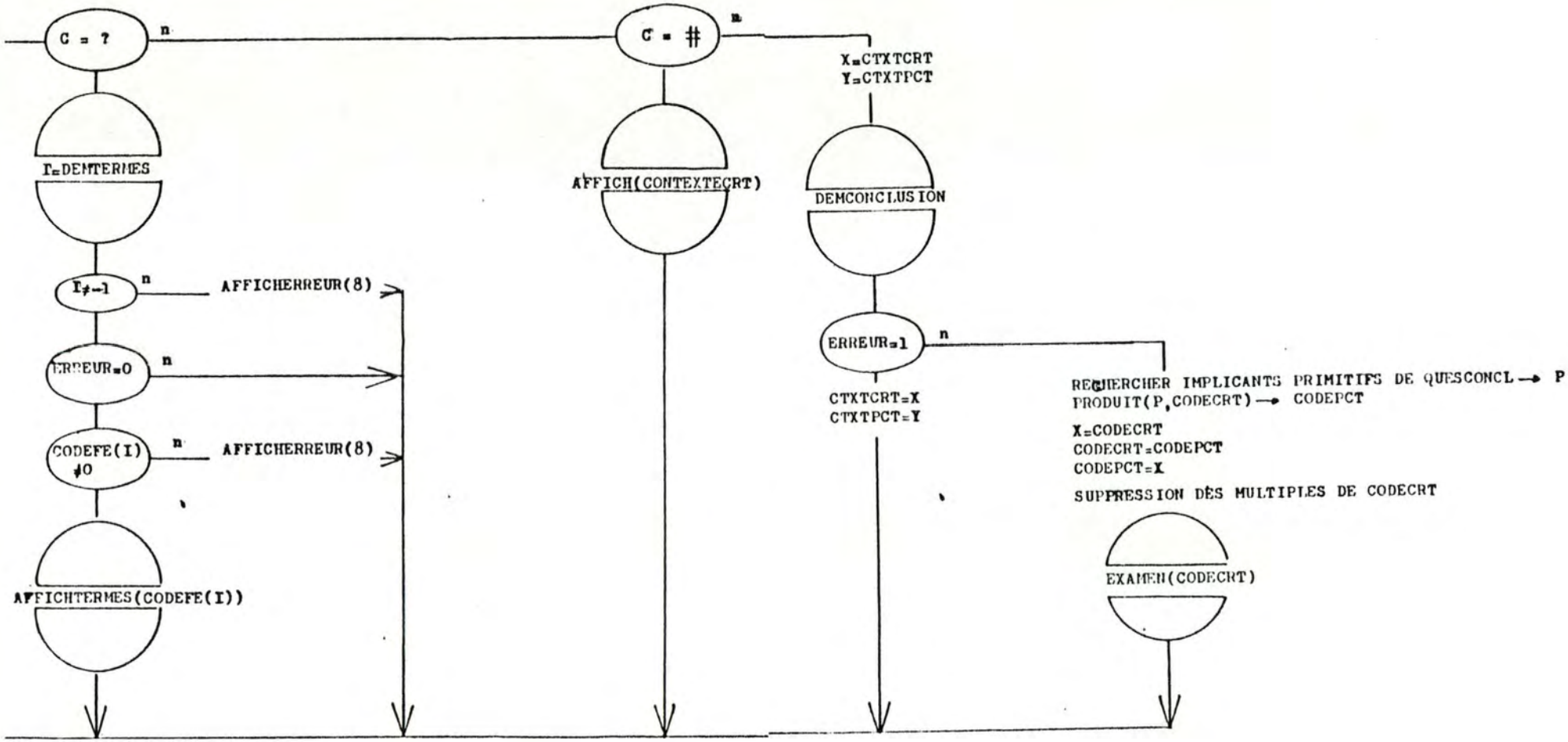
postcond : erreur 3 à 12 suivant le cas (message suffisamment explicite);
ou
d = <return> aucun effet; rien n'est affiché à l'écran.
d = <tab> aucun effet; rien n'est affiché à l'écran.
d = '*' affichage du code courant.
d = ':' retour au contexte initial; aucune réponse à l'écran.
d = '#' affichage du contexte courant.
d = '<' retour au dernier contexte; aucune réponse à l'écran.
d = '$' sortie du programme.
d = <demande de termes>: affichage des termes répondant à la question.
d = <demande d'elt>: affichage des éléments répondant à la question.

d =<demande de conclusion>: affichage de la conclusion à l'écran.
(affichage des termes de la conclusion mis en évidence; si aucun terme ne peut être mis en évidence, affichage de '-').

- si l'antécédent de la question est une tautologie et si le code généré par le compilateur est un vecteur de bits 1 (toutes les prémisses sont des tautologies) alors affichage du message 'TAUTOLOGIE'.
- si l'antécédent de la question est une tautologie et si le code généré par le compilateur n'est pas celui d'une tautologie, alors affichage du code courant.
- si l'antécédent de la question est une fnd nulle, alors affichage du message 'IMPOSSIBLE'.
- si l'antécédent de la question est une fnd en contradiction avec le code courant, alors affichage du message 'IMPOSSIBLE'.


MAIN
DEMANDER NOM DU TEXTE
OUVRIR FICHIER CODE DE CE TEXTE
FICHIER EXISTE
n
PRINTF(ERREUR 1)
CHARGER LE FICHIER CODE
n
MAXSIZE=
MAXFORMAT
PRINTF(ERREUR)
RECOPIER CODE_INITIAL DANS ZONE1
CODECRT = ZONE1
CODEPCT = ZONE2
CONTEXTECRT(0) = 0
CTXTCRT = CTXTPCT = CONTEXTECRT + 1
LIRE C
ERREUR = 0
C = $
n
ANALYSE


ANALYSE
C = RETURN
n
C = TAB
n
C = *
n
C = !
n
C = <
n
C = >
n
AFFICHTERMES (CODECRT)
RECOPIER CODEINITIAL DANS ZONE1
CODECRT = ZONE1
CODEPCT = ZONE2
CONTEXTECRT(0) = 0
CTXTCRT=CTXTPCT = CONTEXTECRT+1
X=CODECRT
CODECRT=CODEPCT
CODEPCT=X
CTXTCRT=CTXTPCT
DEMELT
ERREUR=0
n
POUR I DE 0 A 24*MAXFORMAT
REGARDER SI CODEFE(I)&QUESELT = CODEFE(I)
SI OUI, AFFICHER TRANSFELT(I)
WHILE(C≠RETURN)
LIRE C


C = ?
n
I=DEMTERMES
I≠-1
n
AFFICHERREUR(8)
ERREUR=0
n
CODEFE(I) ≠0
n
AFFICHERREUR(8)
AFFICHTERMES(CODEFE(I))
C = #
n
AFFICH(CONTEXTECRT)
X=CTXTCRT
Y=CTXTPCT
DEMCONCLUSION
ERREUR=1
n
CTXTCRT=X
CTXTPCT=Y
RECHERCHER IMPLICANTS PRIMITIFS DE QUESCONCL → P
PRODUIT(P,CODECRT) → CODEPCT
X=CODECRT
CODECRT=CODEPCT
CODEPCT=X
SUPPRESSION DES MULTIPLES DE CODECRT
EXAMEN(CODECRT)

demconclusion
+++++++++++++

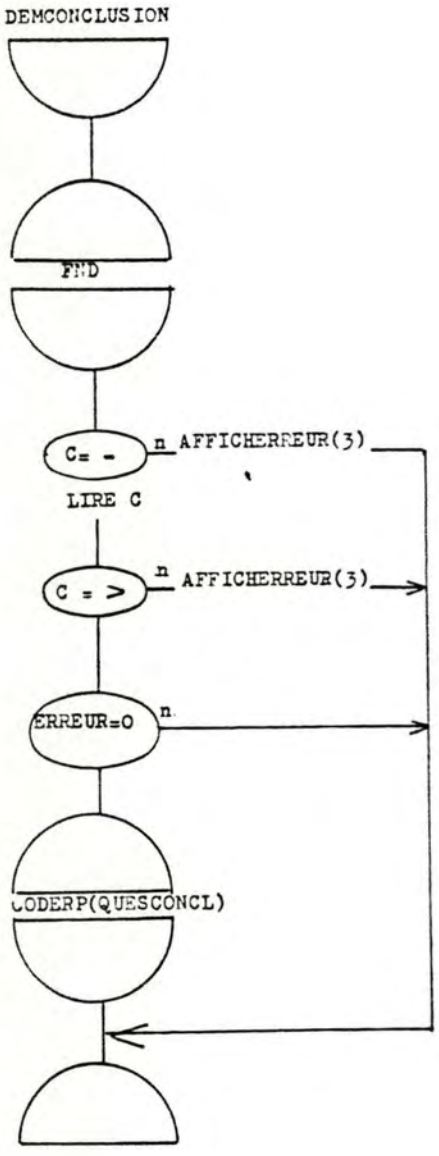
fonction : analyser et coder une question de type demande de conclusion .

argument : une question lue sur l'écran.

precond : il s'agit d'une question de type demande de conclusion

résultat : message d'erreur
ou (exclusif)
code de la question.

postcond : - la question est codée dans quesconcl.
- erreur 3, 4, 5, 9, 10, 11 ou 12, suivant le cas.



demtermes
+++++++++

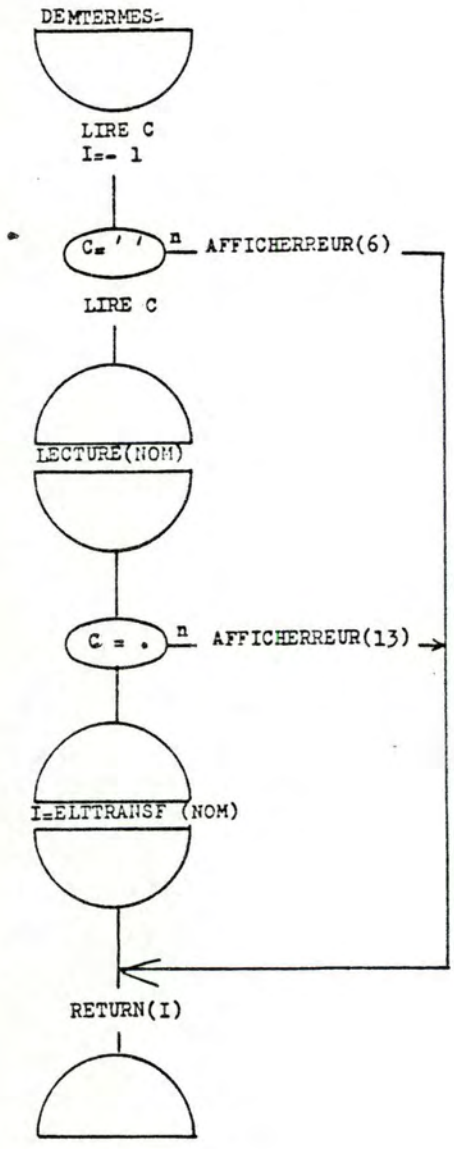
fonction : analyser une question de type demande de termes .

argument : une question lue sur l'écran.

precond : de type demande de termes .

résultat : message d'erreur
ou
un entier i .

postcond : erreur 6 ou 13 suivant le cas.
si erreur alors i = -1 .
si l'élément de la question n'est pas mémorisé : i = -1.
si l'élément de la question est mémorisé, i = indice sous lequel cet élément est mémorisé.



demelt
++++++

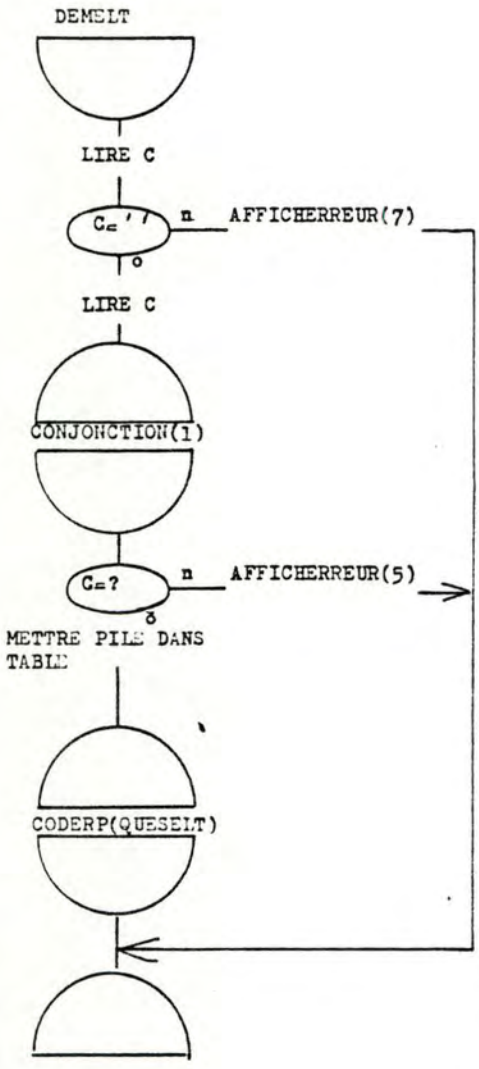
fonction : analyser et coder une question de type demande d'elts .
••••••••

argument : une question lue sur l'écran.
••••••••

précond : de type demande d'elts.
• • • • •

résultat : message d'erreur
••••••••
ou (exclusif)
code de la question.

postcond : la question est codée dans queselt.
• • • • • •
erreur 5, 7, 9, ou 11 suivant le cas.



afficherreur
++++++++++++

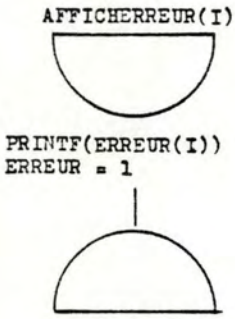
fonction : afficher un message d'erreur.
....

argument : un entier i.
....

précond : i = indice d'un message.
.....

résultat : affichage message d'erreur.
....
erreur = 1.

postcond : message affiché est le message n° i.
......



examen
++++++

fonction : analyser et afficher une forme normale disjonctive.
••••••••

argument : une fnd codé dans un tableau de caractères a .
••••••••

precond : a est terminé par ' 0'.
• • • •

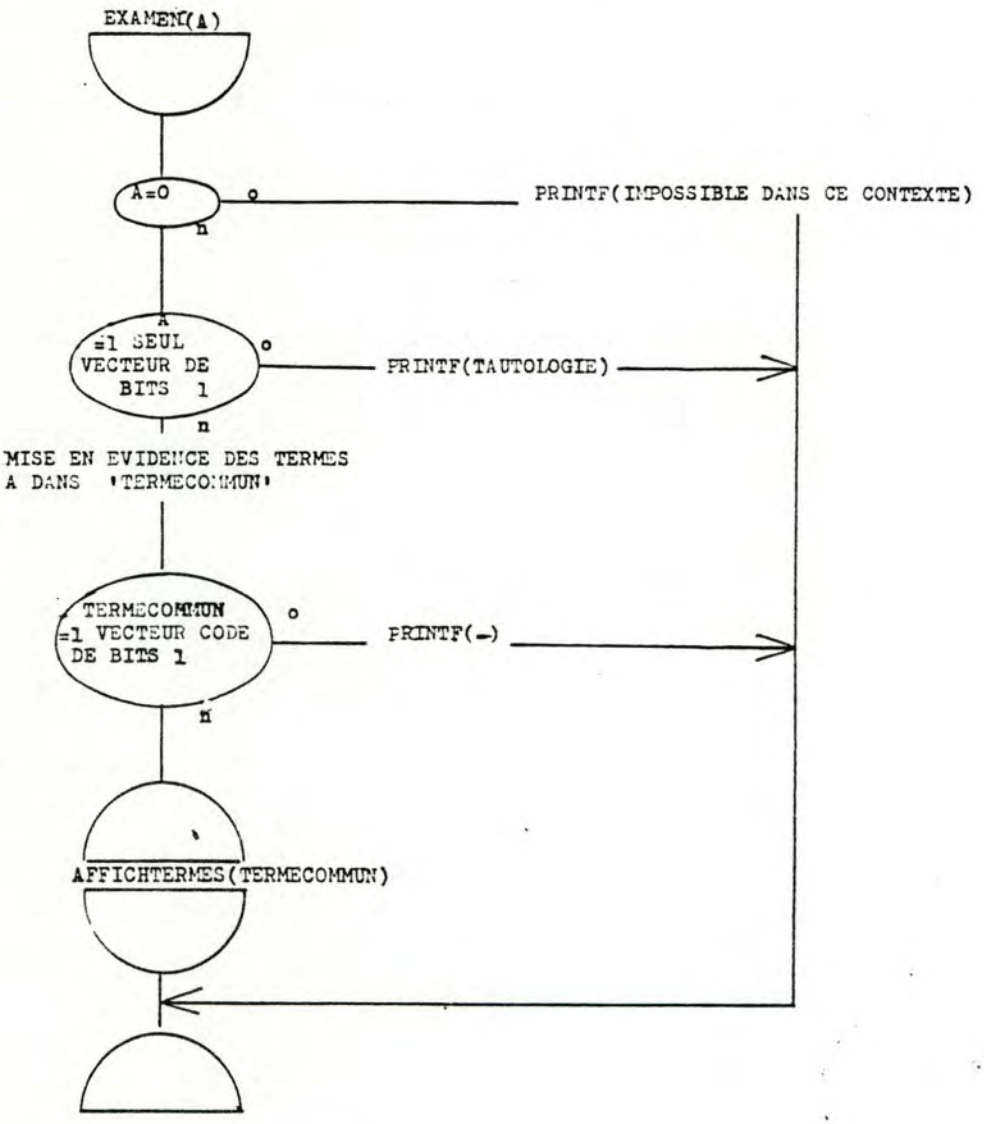
résultat : affichage d'un message.
••••••••

poscond : si fnd = 0 message='IMPOSSIBLE'.

si fnd = 1 message='TAUTOLOGIE'.

si fnd ≠ 0 et ≠ 1 affichage des termes de fnd mis en évidence.

si aucun terme ne peut être mis en évidence , affichage '-'.



affichtermes
++++++++++++

fonction : afficher une forme normale disjonctive fnd .

argument : fnd codé dans un tableau de caractère a .

precond : a est terminé par '\0'.

résultat : affichage de fnd .

postcond : une ',' entre les termes de chaque monôme;
les termes niés sont précédés de '-';
un '+' entre chaque monôme de fnd .

fnd
+++

fonction : analyser une forme normale disjonctive lue sur l'écran.

argument : une fnd lue sur écran.

précond : -

résultat : message d'erreur
ou (exclusif)
une liste dont chaque élément est une liste d'entiers
+ mise à jour de 'contextecrt'.

postcond : message 9, 10, 11, ou 12 suivant le cas.
chaque liste représente un monôme de fnd (propriétés décrites dans conjonction)
'contextecrt' est complété des numéros de chaque terme intervenant dans la fnd (nombre <0 si le terme est nié, >0 si le terme est affirmé); chaque groupe de nombres représente un monôme et est terminé par '\0';
chaque groupe de monômes, représentant une fnd, est terminé par deux '\0'.

conjonction
+++++++++++

fonction : analyser une conjonction lue sur l'écran.

argument : une conjonction lue sur écran + un entier j.

precond : -

résultat : message d'erreur
ou (exclusif)
liste d'entiers i + mise à jour de 'contextecrt'.

postcond : erreur 9, 11 ou 12 suivant le cas.
chaque nombre i représente un terme.
|i| = numéro du terme (attribué par transf),
i>0 si le terme est affirmé,
i<0 si le terme est nié .

ssi j = 0, 'contextecrt' est complété des numéros de chaque terme de la conjonction (nombre<0 si terme nié, >0 si terme affirmé); le groupe de nombre est terminé par '\0'.

mot
+++

fonction : analyser un terme lu sur écran.

argument : un terme lu sur écran.

precond : -

résultat : message d'erreur
ou (exclusif)
un entier i.

poscond : message 9 ou
|i| = numéro du terme (attribué par transf),
i>0 si le terme est affirmé,
i<0 si le terme est nié .

transf
++++++

fonction : rechercher le numéro d'un terme.

argument : une chaîne de caractères d, représentant un terme.

precond : d terminé par '\0'.

résultat : un entier i .

postcond : si le terme a été mémorisé dans transfterme :
i = indice du terme mémorisé + 1
(i est donc toujours différent de 0).
si le terme n'a pas été mémorisé
i = -1 .

lecture
+++++++

fonction : lire une chaîne de caractère sur écran.
••••••••

arguments : 1 chaîne de caractères d,
••••••••••
1 chaîne de caractères e lue sur écran.

precond : d est vide.
• • • • •

résultat : d est remplie des caractères de e .
••••••••

poscond : sont mis dans d les 9 premiers caractères de e ;
• • • • •
le dernier caractère de d est '\0';
la chaîne e est lue jusqu'au premier caractère admis
(celui-ci n'est pas enregistré dans d);
les caractères admis sont : sp
a → z
A → Z
0 → 9 .

coderp
++++++

fonction : coder une forme normale disjonctive F.
••••••••

arguments : tableau de caractères f,
••••••••••
un pointeur g vers caractère ,
F est représentée par une liste dont chaque élément
est une liste d'entiers (fournie par fnd).

précond : g pointe vers le dernier caractère de f; f est vide.
• • • • •

résultat : message d'erreur
••••••••
ou (exclusif)
F est codée.

postcond : F est codée dans le tableau f
• • • • • •
erreur 12 si pas assez de place pour coder f

elttransf
+++++++++

fonction : rechercher le numéro d'un élément.
argument : une chaîne de caractères d, contenant un élément
precond : terminé par '\0'
résultat : un entier i.
postcond : si l'élément est mémorisé :
i = indice sous lequel l'élément est mémorisé dans transfelt.
si non :
i = -1.

affich.
++++++

fonction : afficher sur écran une suite de fnd.
argument : un tableau a de caractères, contenant le code des fnd à afficher.
precond : ce tableau contient une suite de nombre i;
chaque groupe de nombre représente un monôme et est terminé par '\0';
chaque groupe de monôme représente une fnd et est terminé par deux '\0';
les nombres > 0 représentent des termes affirmés;
les nombres < 0 représentent des termes niés;
|i| = indice sous lequel le terme est mémorisé dans transfterme.
résultat : les fnd codées sont affichées.
postcond : chaque fnd est séparée des autres par le symbole '—>'
si a ne contient que le caractère '\0' : affichage du message 'CONTEXTE INITIAL'.

6. Exemples

1. Soient les prémisses suivantes :

```
animal , -vertebre (eponge );
amphib (crapaud );

› animal + vegetal -› mortel ;
› poisson + amphib + reptile + oiseau + mammif -› vertebre ;
› poisson + amphib + reptile + oiseau -› -mammif ;

poisson (carpe , raie );
oiseau (pie );
reptile (tortue , serpent );

› primate + rongeur + cetace -› mammif ;

rongeur (rat );
› dauphin -› cetace ;
› singe + homme -› primate ;
› singe + rongeur + cetace -› -homme ;
› vertebre -› animal ;
› lichen -› vegetal ;

› (x) dieu + vegetal + animal ;

› dieu -› -vegetal , -animal ;

dauphin (flipper );
dieu (zeus , neptune );
poisson (neptune );

› grec + romain -› homme ;

romain (neron );
grec (socrate , aristote );
-sage (neron );
sage (aristote );

› animal -› mammif , poilu + -mammif , -poilu ;
› animal -› pondeur , -mammif + -pondeur , mammif ;
› animal , -vertebre + homme , -sage -› primitif , -homme + primitif , mauvais
$
```

La compilation envoie le message :
"L'élément suivant ne peut exister : neptune"

Voici un exemple d'exécution

```
> mammif ?
  rat, flipper, neron, socrate, aristote.
> homme ?
  neron, socrate, aristote
> mammif, -homme ?
  rat, flipper
> dieu ?
  zeus
> animal ?
  crapaud, carpe, raie, pie, tortue, serpent, rat, flipper,
  neron, socrate, aristote
> animal,pondeur ?
  crapaud, carpe, raie, pie, tortue, serpent, éponge
> grec, sage ?
  aristote
> homme, mauvais ?
  néron
> primitif ?
  éponge, néron
? socrate.
vertebre, animal, mortel, -cetace, -rongeur, primate, mammif,
homme, -singe, -dauphin, grec, -dieu, -pondeur, poilu.
vertébré->
          vertebre, animal, mortel, -dieu
          mammif->
                  vertebre, animal, morte, -dieu, mammif,
                  -pondeur, poilu
:
mammif, pondeur->
          IMPOSSIBLE DANS CE CONTEXTE
pondeur, poilu->
          IMPOSSIBLE DANS CE CONTEXTE
primate->
     vertebre, animal, mortel, primate, mammif, -dieu, -pondeur,
     poilu
oiseau + poisson->
     vertebre, animal, mortel, -dieu

etc....
```

2. Un conducteur exprime ses opinions sur les automobiles :
   - une traction-avant tient bien la route.
   - il est indispensable qu'une voiture lourde ait de bons freins.
   - toute voiture de grande puissance est d'un prix élevé.
   - les voitures légères n'ont pas une bonne tenue de route.
   - une voiture de faible puissance ne peut pas avoir de bons freins.

   Acceptera-t-il un traction-avant bon marché ?

   Le programme :

```
> t avant -> tb route ;
> tb route -> t avant ;
> v lourde -> b freins ;
> puissant -> cher ;
> -v lourde -> -tb route ;
> -puissant -> -b freins
$
```

   Exécution :

```
 t avant, -cher->
      IMPOSSIBLE DANS CE CONTEXTE
```

3. Les caractéristiques de deux types d'avions sont résumées dans la suite de propositions suivantes en remarquant qu'elles ont été émises de différentes sources et risquent d'être sujettes à caution.
   a) un appareil avec un moteur à réaction et un faible rayon d'action est un bombardier.
   b) les bombardiers à moteur conventionnel ont un armement lourd.
   c) les chasseurs à moteur conventionnel ont un court rayon d'action.
   d) les appareils à moteur conventionnel et long rayon d'action ont un armement léger.
   e) les appareils à réaction ont un armement lourd.
   f) les appareils à armement lourd et à court rayon d'action sont des chasseurs.

g) les appareils à grand rayon d'action ou de chasse ont un armement léger.

h) les appareils à moteur conventionnel ou à rayon limité ont un armement lourd.

Il s'agit de vérifier si ces propositions sont compatibles et dans la négative, d'éliminer de cet ensemble l'une ou l'autre des propositions de façon à pouvoir tirer une conclusion valide et cohérente.

Le programme :

```
> bombard -> -chasseur ;
> -chasseur -> bombard ;
> conventio -> -reaction ;
> -reaction -> conventio ;
> r act red -> -r act et ;
> -r act et -> r act red ;
> armlourd -> -armleger ;
> -armleger -> armlourd ;
> reaction , r act red -> bombard ;
> bombard , conventio -> armlourd ;
> chasseur , conventio -> r act red ;
> conventio , r act red -> armleger ;
> reaction -> armlourd ;
> armlourd , r act red -> chasseur ;
> r act et + chasseur -> armleger ;
> conventio + r act red -> armlourd
$
```

La compilation:

"il existe une prémisse en contradiction avec les autres :16"

Après suppression de la seizième prémisse, l'exécution donne :

*

- reaction, conventio, chasseur, -bombard, armleger, -armlourd,
- r act et, r act red.

4. Trois Secrétaires d'Etat, MM. Artelin, Bertrand et Cochet, appartenant tous à un même parti, risquent d'être impliquées, à des degrés divers, dans des affaires politico-financières, dont l'une a trait à l'installation d'une raffinerie de pétrole (affaire R), la seconde à la construction de supermarchés (affaire S), et la troisième à des fournitures de matériel téléphonique (affaire T).

On s'attend à ce qu'éclatent un ou plusieurs scandales à propos de ces affaires; en conséquence, les dirigeants du

parti se préparent à sacrifier, si besoin en est, l'un ou l'autre bouc émissaire, choisis parmi des personnalités d'importance secondaire, dont les trois Secrétaires d'Etat.

Les décisions du parti peuvent se résumer comme suit :

- Personne ne démissionnera si aucun scandale public n'éclate.
- Bertrand démissionnera si et seulement si les trois scandales éclatent, et sa démission entraînera celle de Cochet.
- Cochet démissionnera aussi si un seul scandale éclate; il restera cependant en fonction si deux scandales seulement éclatent, car alors la hache devra frapper plus haut, et le sacrifice de Cochet serait inutile.
- Artelin démissionnera si et seulement si le scandale des téléphones vient à éclater.
- Le scandale des supermarchés impliquera la démission de de Cochet ou bien celle de Bertrand; mais si cette affaire n'éclate pas, la démission éventuelle de Cochet impliquera celle de Bertrand, et vice versa.

Dites quelles seront les démissions, pour chacun des 8 cas de scandales a priori possibles.

Le programme :

```
> -R , -S , -T -> -Artelin , -Bertrand , -Cochet ;

> R , S , T -> Bertrand ;
> Bertrand -> R , S , T ;
> Bertrand -> Cochet ;

> R , -S , -T + -R , S , -T + -R , -S , T -> Cochet ;
> R , S , -T + R , -S , T + -R , S , T -> -Cochet ;

> T -> Artelin ;
> Artelin -> T ;

> S -> Cochet + Bertrand ;
> -S -> -Cochet + Bertrand ;
> -S -> -Bertrand + Cochet
$
```

L'exécution :

R, S, T ⟶
R, S, T, Artelin, Cochet, Bertrand

:

-R, -S, -T ⟶
-R, -S, -T, -Artelin, -Cochet, -Bertrand

:

R, -S, -T ⟶
IMPOSSIBLE DANS CE CONTEXTE

:

-R, S, -T ⟶
-R, S, -T, -Artelin, -Bertrand, Cochet

:

-R, -S, T ⟶
IMPOSSIBLE DANS CE CONTEXTE

:

R, S, -T ⟶
IMPOSSIBLE DANS CE CONTEXTE

:

R, -S, T ⟶
R, -S, T, Artelin,-Bertrand, -Cochet

:

-R, S, T ⟶
IMPOSSIBLE DANS CE CONTEXTE

:

Artelin ⟶
T, R, Artelin

Bertrand ⟶
S, T, R, Artelin, Bertrand, Cochet

:

Cochet ⟶
S, Cochet

*
S, T, R, Artelin, Bertrand, Cochet + S, -T, -R,
-Artelin, -Bertrand, Cochet

etc ....

5. A la veille des élections législatives, trois partis politiques, les Chrétiens (K), les Libéraux (L) et les Marxistes (M), décident de former une coalition gouvernementale, pour autant qu'ils obtiennent ensemble la majorité des sièges au Parlement.

   Au sein de cette coalition, seront déclarés "prépondérants" le ou les partis qui auront obtenu le plus de sièges : on pourra donc avoir soit 1, soit 2, soit 3 partis prépondérants, et même aucun prépondérant si et seulement si l'ensemble n'a pas obtenu la majorité au Parlement.

   Parmi les ministrables, on cite le général en retraite Auclair, le syndicaliste Bouillon et le chanoine Calaux, à propos desquels les partis ont passé la convention ci-après :

   Art. 1 - Auclair sera ministre si et seulement si les Libéraux sont prépondérants.

   Art. 2 - Calaux sera ministre si les Chrétiens sont prépondérants, mais à condition que les Libéraux ne le soient pas.

   Art. 3 - Si les Chrétiens ne sont pas prépondérants, Calaux sera ministre si et seulement si Auclair l'est également.
   Art. 4 - Bouillon sera ministre si les Marxistes sont prépondérants, mais à condition que Calaux ne soit pas ministre.

   Art. 5 - Si les Marxistes ne sont pas prépondérants, Calaux sera ministre si et seulement si Bouillon l'est aussi.
   Quelles sont les combinaisons ministérielles possibles pour ces trois politiciens, en fonction des résultats électoraux possibles ?

   Le programme :

```
> Auclair -> L ;
> L -> Auclair ;

> -L , K -> Calaux ;

> -K -> -Calaux , -Auclair + Calaux , Auclair ;

> M , -Calaux -> Bouillon ;

> -M -> Calaux , Bouillon + -Bouillon , -Calaux
$
```

L'exécution :

-K, -L, -M ⟶

-K, -L, -M, -Auclair, -Bouillon, -Calaux

:

K, L, M ⟶

K, L, M, Auclair

:

-K, L, M ⟶

-K, L, M, Auclair

:

K, -L, M ⟶

K, -L, M, -Auclair, Calaux

:

K, L, -M ⟶

K, L, -M, Auclair

:

-K, -L, M ⟶

-K, -L, m, -Auclair, Bouillon, - Calaux

:

K, -L, -M ⟶

K, -L, -M, -Auclair, Bouillon, Calaux

:

-K, L, -M ⟶

-K, L, -M, Auclair, Bouillon, Calaux

:

Auclair ⟶

L, Auclair

etc ....

6. Dans une entreprise, 4 types de primes peuvent être octroyés aux employés s'ils remplissent les conditions suivantes :

   1. prime d'ancienneté - (PA)
      Si l'employé possède au moins 3 années de service (a_3) et si son absence au bureau est inférieure ou égale en moyenne à 1 mois ou 4 semaines (b_4) la prime PA s'élève à 15 % du salaire mensuel (S_m).

   2. prime de présence - (PP)
      Si l'employé a une ancienneté d'au moins un an (a_1) et si son absence est limitée à 2 semaines par an (b_2) la prime (PP_1) s'élève à 20 % de S_m diminuée de 8 % par semaine d'absence.

      Si l'employé a une ancienneté inférieure à un an (a_1), il recevra une prime de présence (PP_2) de 6 % du S_m pour autant que son absence moyenne soit inférieure à 2 jours(b_o).

   3. prime de productivité - (PT)
      Si l'employé a une ancienneté d'au moins 3 ans (a_3) et s'il a été absent moins de 2 semaines (b_2) ou si dans le cas contraire, ayant été absent moins d'un mois (b_4), son signalement (n) est supérieur à 15, la prime (PT_1) vaudra $\frac{n}{20}$ x 20 % de S_m.
      Si l'employé a une ancienneté inférieure à 3 ans (a_3) mais supérieure à 1 an (a_1), il reçoit une prime (PT_2) de $\frac{n}{20}$ X 10 % S_m pour autant que son absence soit inférieure à 2 semaines (b_2).

      Si l'employé a une ancienneté inférieur à 1 an (a_1), son absence est limitée à 2 jours (b_o) et si son signalement est supérieur à 15 sa prime (PT_3) s'élèvera à (n - 15) X 2 % de S_m.

   4. prime pour service exceptionnel - (PS)
      Si l'employé a le minimum d'absence (b_o) et possède une cote de signalement supérieure à 15 la prime s'élève à (n - 15) X 4 % de S_m.

On trouve les incompatibilités :

$a_3\ a_1$

et $(b_4\ b_2 + b_4\ b_o + b_2\ b_o)$

d'où le programme :

```
> a3 , b4 -> PA ;
> a1 , b2 -> PP1 ;
> -a1 , b0 -> PP2 ;
> a3 , b2 + a3 , b4 , n -> PT1 ;
> a1 , -a3 , b2 -> PT2 ;
> -a1 , b0 , n -> PT3 ;
> b0 , n -> PS ;

> a3 -> a1 ;
> (x) b4 , b2 + b4 , -b0 + -b2 , -b0
$
```

L'exécution :

a_3, $-a_1$, b_4 $\Longrightarrow$

IMPOSSIBLE DANS CE CONTEXTE

a_3, a_1, b_4, $-b_2$, $-b_o$ $\Longrightarrow$

PA .

n $\Rightarrow$ PA, PT_1

Le programme est tel qu'à la question "fnd $\Rightarrow$"

- la réponse est "IMPOSSIBLE DANS CE CONTEXTE" si fnd contient des incompatibilités
- la réponse ne contient que des variables affirmées (toutes les autres sont supposées niées)

On a programmé ainsi la table de décision suivante :

	I	II	III	IV	V	VI	VII	VIII	IX	X
a_3	0	0	0	0	0	1	1	1	1	1
a_1	0	0	1	1	1	1	1	1	1	1
b_4	1	1	1	1	1	1	1	1	1	1
b_2	1	1	1	1	1	0	0	1	1	1
b_o	1	1	0	1	1	0	0	0	1	1
n	0	1	-	0	1	0	1	-	0	1
PA	0	0	0	0	0	1	1	1	1	1
PP_1	0	0	1	1	1	0	0	1	1	1
PP_2	1	1	0	0	0	0	0	0	0	0
PT_1	0	0	0	0	0	0	1	1	1	1
PT_2	0	0	1	1	1	0	0	0	0	0
PT_3	0	1	0	0	0	0	0	0	0	0
PS	0	1	0	0	1	0	0	0	0	1

rem : si la question "fnd $\Rightarrow$" traite un cas non repris dans la table, alors la réponse ne contient pas d'informations nouvelles (réponse = fnd).

7. Après le calcul du montant brut d'une facture, une réduction éventuelle doit être établie.
   Les règles de ce calcul sont les suivantes :

   1. si le client est de catégorie 1 (C_1), il obtient une réduction A(R_A) s'il transporte les marchandises par ses propres moyens (p) ou si ses habitudes de paiement (h) sont bonnes ou si le montant brut de la facture est supérieur à 50 000 F (50). Si deux de ces conditions sont satisfaites, il reçoit les réductions A, B et C (R_C).
   2. si le client n'appartient pas à la catégorie 1 (C_1), il obtient une réduction A si le montant brut de la facture est supérieur à 60 000 F (60) et s'il a transporté les marchandises par ses propres moyens.
   3. si le client n'est pas de catégorie 1 et si le montant brut n'est pas supérieur à 50 000 F, il y a erreur (D) dans les données.

Le programme :

```
> 60 -> 50 ;
> C1 , p , -h , -50 + C1 , -p , h , -50 + C1 , -p , -h , 50 + -C1 , 60 , p-> Ra ;
> C1 , p , h , -50 + C1 , p , -h , 50 + C1 , -p , h , 50 -> Rb ;
> C1 , p , h , 50 -> Rc ;

> -C1 , -50 -> D
$
```

donne la table de décision :

$C \cdot p \cdot h$ / 50 · 60 / Cas	I	II	III	IV	V	VI	VII	VIII	IX
C_1	0	0	1	1	1	1	1	1	1
p	–	1	0	0	0	1	1	1	1
h	–	–	0	1	1	0	0	1	1
50	0	1	1	0	1	0	1	0	1
60	0	1	–	0	–	0	–	0	–
R_A	0	1	1	1	0	1	0	0	0
R_B	0	0	0	0	1	0	1	1	0
R_C	0	0	0	0	0	0	0	0	1
D	1	0	0	0	0	0	0	0	0
Nb. cas	4	2	2	1	2	1	2	1	2

```
# include <stdio.h>
# define MAXSIZE 10
# define placereservee MAXSIZE*200

/* MAXSIZE definit le nombre de caracteres utilises pour coder
   un monome;
   un caractere possedant 8 bits, le nombre maximum de termes
   memorisables est 4*MAXSIZE;
*/



FILE *fichier,*fopen( );

/* 'fichier' contient le texte a analyser et coder;
*/



int c,erreur,maxterme,maxelt,numpremisse;

/* c est le caractere courant, lu dans 'fichier';

   erreur = 0 si aucune erreur n'est rencontree dans texte;
   erreur = 1 des qu'une erreur est rencontree dans le texte;

   maxterme est le nombre courant de termes memorises;
   maxelt est le nombre courant d'elements memorises;

   numpremisse est le numero courant de la premisse analysee;
*/



char codePE[MAXSIZE*24][MAXSIZE+1],
     zonetravail[placereservee + 1],
     codeFU[placereservee + 1],
     pourtoutx[placereservee + 1];

/* codePE contient le code des premisses existentielles;
   codeFU contient le code des premisses universelles;
   pourtoutx contient le code de la premisse universelle courante;
*/



char transfterme[4*MAXSIZE][10],
     transfelt[24*MAXSIZE][10];

/* transfterme contient les termes memorises;
   transfelt contient les elements memorises;
*/



char *msgerreur[15] =
     {"ce texte n'existe pas\n",
      "fin de texte introuvable\n",
      "en tete de la premisse universelle [(x) ]: ",
      "en tete de la premisse existentielle : ",
      "forme implication non correcte : ",
       "fin de la premisse existentielle : ",
      "trop de termes employes: MAXSIZE * 4 au max\n",
      "trop d'elements employes: MAXSIZE * 24 au max\n",
      "pas assez de place: modifier placereservee\n",
```

```
      "attention; un blanc apres le symbole '+' : ",
      "attention; un blanc apres le symbole ',' : ",
      "attention; un blanc apres le symbole '>' : ",
      "il existe un premisse en contradiction avec les autres: ",
      "l'ensemble des premisses universelles forme une tautologie.\n",
      "l'element suivant ne peut exister : "
      };


struct nbre
         {int nb;
          struct nbre *next;} *pile;

struct tableau
         {struct nbre *pt;
          struct tableau *suivant} *table;

/* table et pile contiennent le resultat de l'analyse d'une fnd;
*/



main( )

/* assurer la coordination entre les procedures dans le but
   d'analyser et coder le texte de 'fichier';
*/

{
char x,a,e,f,
     titrein[10],titreout[12],
     termecommun[MAXSIZE],t[MAXSIZE + 1];
char anccons[2*placereservee + 1],
     nveaucons[2*placereservee + 1];
char *p,*q,*r,*rechimplicants( ),*y,*TEXTE( );
int i,j,l;

erreur=1;
while(erreur==1)
           {
           i=0;
           printf("introduire le nom du texte: \n");
           while(((x=getchar( ))!='\n')&&(i!=9))
                  titrein[i++] = x;
           while(x!='\n')
                  x=getchar( );
           titrein[i] = '\0';

           if ((fichier=fopen(titrein,"r"))==NULL)
                  {
                  printf(msgerreur[0]);
                  printf("\n");
                  }
              else
                  erreur=0;
           }

numpremisse=0;
maxterme=maxelt=0;

y = TEXTE( );

if (c!='$')
      printf (msgerreur[1]);
fclose(fichier);
```

```
if (erreur == 0)
        {
        l = (2*placereservee) + 1;
        p = rechimplicants(codePU,y,anccons,nveaucons,l);
        for(i=0;i!=MAXSIZE;i++)
              t[i] = ~0;
        t[i] = '\0';

        j = 0;
        i = 0;
        while(comparaison(&codePE[j][i],t)!=0)
                {
                q = zonetravail;
                r = zonetravail +(placereservee + 1);
                produit(codePE[j],p,q,&r);
                if (*q=='\0')
                        {
                        printf(msgerreur[14]);
                        printf(transfelt[j]);
                        printf("\n");
                        for (i=0;i!=MAXSIZE;i++)
                                codePE[j][i] = ~0;
                        codePE[j][i] = '\0';
                        }
                    else
                        {
                        for(i=0;i!=MAXSIZE;i++)
                              termecommun[i] = *q++;
                        while(*q != '\0')
                                {
                                for(i=0;i!=MAXSIZE;i++)
                                      termecommun[i] = termecommun[i] | *q++;
                                }
                        for(i=0;i!=MAXSIZE;i++)
                              codePE[j][i] = termecommun[i];
                        codePE[j][i] = '\0';
                        }
                j++;
                i = 0;
                }
        }

if (erreur==0)
        {
        i = 0;
        while(titrein[i]!='\0')
                {
                titreout[i] = titrein[i];
                i++;
                }
        titreout[i++] = '.';
        titreout[i++] = 'o';
        titreout[i] = '\0';
        fichier = fopen(titreout,"w");

        putc(MAXSIZE,fichier);

        for(i=0;i!=MAXSIZE*24;i++)
                {
                for(j=0;j!=MAXSIZE+1;j++)
                      putc(codePE[i][j],fichier);
                }

        while (*p!='\0')
                putc(*p++,fichier);
        x = '\0';
```

```
        putc(x,fichier);

        for(i=0;i!=4*MAXSIZE;i++)
                {
                for(j=0;j!=10;j++)
                        putc(transfterme[i][j],fichier);
                }

        for(i=0;i!=24*MAXSIZE;i++)
                {
                for(j=0;j!=10;j++)
                        putc(transfelt[i][j],fichier);
                }
        fclose(fichier);
        }

printf("FIN\n");
}


lirecar(c)
int c;

/* lire le caracter suivant de 'fichier';
*/

{
if (c!=EOF)
          {
          while((c=getc(fichier))=='\n'||c=='\t');
          }

return(c);
}


char *TEXTE( )

/* analyser et coder le texte de 'fichier';
*/

{
int i,j;
char *z,*PREMISSE( );

for(i=0;i!=MAXSIZE;i++)
        codePU[i] = ~0;
codePU[i] = '\0';
pourtoutx[0] = '\0';
zonetravail[0] = '\0';

for(i=0;i!=4*MAXSIZE;i++)
        {
        for(j=0;j!=MAXSIZE;j++)
                transfterme[i][j] = ~0;
        transfterme[i][j] = '\0';
        }

for(i=0;i!=24*MAXSIZE;i++)
        {
        for(j=0;j!=MAXSIZE;j++)
                transfelt[i][j] = ~0;
        transfelt[i][j] = '\0';
        }

for(i=0;i!=MAXSIZE*24;i++)
```

```
        {
        for(j=0;j!=MAXSIZE;j++)
                codePE[i][j] = '0';
        codePE[i][j] = '\0';
        }

c = ';';
z = &codePU[MAXSIZE];

while (c == ';')
        z = PREMISSE(z);

return(z);
}


char *PREMISSE(z)
char *z;

/* analyser et coder la premisse courante;
*/

{
char *g,x,*y;

x = '\0';
g = &x;
numpremisse++;

c=lirecar(c);
if (c == '>')
        {
        c = lirecar(c);
        if (c != ' ')
                {
                printf(msgerreur[11]);
                printf("%d\n",numpremisse);
                erreur = 1;
                }
        FORMEU();
        if (erreur == 0)
                {
                y = zonetravail + (placereservee + 1);
                z = codePU + (placereservee + 1);
                produit(codePU,pourtoutx,zonetravail,&y);
                suppressionmltple(zonetravail,g,codePU,&z);
                if (codePU[0] == '\0')
                        {
                        printf (msgerreur[12]);
                        printf ("%d\n",numpremisse);
                        erreur = 1;
                        }
                }
        }
   else
        FORMEE();

return(z);
}


FORMEI()

/* analyser et coder une implication;
*/
```

```
{
char *p,*q,*h;

FND( );

if (c!='-')
        {
        printf(msgerreur[4]);
        printf("%d\n", numpremisse);
        erreur=1;
        }

c=lirecar(c);
if (c!='>')
        {
        printf(msgerreur[4]);
        printf("%d\n", numpremisse);
        erreur=1;
        }

if (erreur==0)
        {
        p = pourtoutx + (placereservee + 1);
        codern(pourtoutx,&p);
        }

c=lirecar(c);
if (c != ' ')
        {
        printf(msgerreur[11]);
        printf("%d\n",numpremisse);
        erreur = 1;
        }
     else
        c =lirecar(c);

FND( );

if (erreur==0)
        {
        q = pourtoutx + (placereservee + 1);
        coderp(p,&q);
        }
}


FORMEU( )

/* analyser et coder un premisse universelle;
*/

{
char *r;

c=lirecar(c);
if (c=='(')
        {
        c=lirecar(c);
        if (c!='x')
                {
                printf(msgerreur[2]);
                printf("%d\n", numpremisse);
                erreur=1;
                }
        c=lirecar(c);
```

```
        if (c!=')')
                {
                printf(msgerreur[2]);
                printf("%d\n", numpremisse);
                erreur=1;
                }
        c=lirecar(c);
        if (c != ' ')
                {
                printf(msgerreur[2]);
                printf("%d\n",numpremisse);
                erreur = 1;
                }
            else
                c = lirecar(c);

        FND();

        if (erreur==0)
                {
                r = pourtoutx + (placereservee + 1);
                coderp(pourtoutx,&r);
                }
        }
    else
        FORMEI();
}


FND()

/* analyser et coder dans (table et pile) une fnd;
*/

{
struct tableau *t,*calloc();
int i;

table = NULL;

i=0;
while(i==0)
        {
        CONJONCTION();

        if (erreur==0)
                {
                t=calloc(1,sizeof(struct tableau));
                t->pt=pile;
                t->suivant=table;
                table=t;
                }

        if(c=='+')
                {
                c=lirecar(c);
                if ( c!= ' ')
                        {
                        printf(msgerreur[9]);
                        printf("%d\n",numpremisse);
                        erreur = 1;
                        }
                    else
                        c = lirecar(c);
                }
```

```
            else
                i=(-1);
        }
}


CONJONCTION( )

/* analyser et coder dans (pile) une conjonction de termes;
*/

{
struct nbre *p,*calloc( );
int i,nombre;

i=0;
pile=NULL;
while (i==0)
        {
        nombre=MOT( );

        if (erreur==0)
                {
                p=calloc(1,sizeof(struct nbre));
                p->nb=nombre;
                p->next=pile;
                pile=p;
                }

        if (c==',')
                {
                c=lirecar(c);
                if (c != ' ')
                        {
                        printf(msgerreur[10]);
                        printf("%d\n",numpremisse);
                        erreur = 1;
                        }
                     else
                        c = lirecar(c);
                }
            else
                i=(-1);
        }
}


coderp(f,g)
char *f,**g;

/* coder une fnd (table et pile) dans le tableau f;
*/

{
int val;
char T[MAXSIZE];
int j,k,l;

while ((table!=NULL)&&(f!=*g))
        {
        for(j=0;j!=MAXSIZE;j++)
                T[j] = '0';

        while(table->pt != NULL)
```

```
                {
                k=table->pt->nb;
                if (k < 0)
                        k =(-k);
                l = k - 1;
                l = l/4;
                k = k % 4;
                k = 8-(2*k);
                k = k % 8;
                if (table->pt->nb < 0)
                        val = 1 << k;
                    else
                        val = 2 << k;
                val = ~val;
                T[l]=T[l] & val;
                table->pt = table->pt->next;
                }

        for(j=0;((j!=MAXSIZE)&&(f!=g[0]));j++)
                *f++ = T[j];

        table = table->suivant;
       }

if (f==*g)
        {
        erreur = 1;
        printf(msgerreur[8]);
        }
*f = '\0';
*g = f;

}


codern(f,g)
char *f,**g;

/* coder la negation d'une fnd (table et pile) dans le tableau f
*/

{
int j,k,l;
int val;
char T[MAXSIZE],x,*u,r[placereservee+1],
     *v,*s,*p,*q,*maximum;

x ='\0';
u = &x;

p = f;
for(j=0;j!=MAXSIZE;j++)
        *p++ = ~0;
*p = '\0';

s = r;
maximum = s + placereservee + 1;

while ((table!=NULL)&&(erreur==0))
        {
        q = s;

        while ((table->pt!=NULL) && (erreur==0))
                {
                for(j=0;j!=MAXSIZE;j++)
                        T[j]=~0;
```

```
                k = table->pt->nb;
                if (k < 0)
                        k = (-k);
                l = k -1;
                l = l/4;
                k = k%4;
                k = 8-(2*k);
                k = k % 8;
                if (table->pt->nb <0)
                        val = 2 << k;
                    else
                        val = 1 << k;
                val = ~val;
                T[l] = T[l] & val;
                for (j=0;((j!=MAXSIZE)&&(s!=maximum));j++)
                        *s++ = T[j];

                if (s==maximum)
                        {
                        printf(msgerreur[8]);
                        erreur = 1;
                        }

                table->pt = table->pt->next;
                }

        *s = '\0';

        v = zonetravail + (placereservee + 1);
        produit(f,q,zonetravail,&v);

        v = *g;
        suppressionmltple(zonetravail,u,f,&v);

        table = table->suivant;
        }
*g = v;
}


MOT( )

/* analyser et memoriser un terme
*/

{
int i;
char nomdeterme[10];

if (c=='-')
        {
        c=lirecar(c);
        LECTURE(nomdeterme);

        i=transf(nomdeterme);
        i = (-i);
        if (i==1)
                {
                printf(msgerreur[6]);
                printf("%d\n", numpremisse);
                erreur = 1;
                }
        }
    else
        {
        LECTURE(nomdeterme);
```

```
        i=comparaison(transfterme[0],nomdeterme);
        i=transf(nomdeterme);
        if (i==-1)
                {
                printf(msgerreur[6]);
                printf("%d\n", numpremisse);
                erreur = 1;
                }
        }

return(i);
}



LECTURE(d)
char *d;

/* lire une chaine de caractere de 'fichier' dans le vecteur d;
*/

{
int e,f,g,h,i;

i=0;
e = (c==' ');
f = ((c>='0')&&(c<='9'));
g = ((c>='A')&&(c<='Z'));
h = ((c>='a')&&(c<='z'));
while ((e || f || g || h)&&(i!=9))
        {
        d[i++] = c;
        c = lirecar(c);
        e = (c==' ');
        f = ((c>='0')&&(c<='9'));
        g = ((c>='A')&&(c<='Z'));
        h = ((c>='a')&&(c<='z'));
        }
d[i] = '\0';

if (i==9)
        {
        while (e || f || g || h)
                {
                c = lirecar(c);
                e = (c==' ');
                f = ((c>='0')&&(c<='9'));
                g = ((c>='A')&&(c<='Z'));
                h = ((c>='a')&&(c<='z'));
                }
        }

}



transf(d)
char *d;

/* attribuer un numero a un terme;
*/

{
int i,j;

for(i=0;((i!=maxterme)&&(comparaison(transfterme[i],d)!=0));i++);
```

```
if (i== MAXSIZE*4)
        i=(-2);
  else
        {
        if (i==maxterme)
                {
                copie(transfterme[i],d);
                maxterme++;
                }
        }
i++;

return(i);
}



simplification(k)
char *k;

/* determiner si une chaine de caracteres contient le doublet
   de bits '00';
*/

{
int i,j,s,val;

s=0;
for(j=0;j!=MAXSIZE;j++)
        {
        i=0;
        val=1;
        while ((i!=8)&&(val!=0))
                {
                val = (*(k+j) >> i);
                val = val & 3;
                i=i+2;
                }
        if (val==0)
                {
                s=(-1);
                break;
                }
        }

return(s);
}



copie(s1,s2)
char *s1,*s2;

/* copier une chaine de caractere dans une autre;
*/

{
while (*s1++ = *s2++);
}



comparaison(s1,s2)
char *s1,*s2;

/* comparer deux chaines de caracteres;
```

```
*/

{
for(;*s1==*s2;s1++,s2++)
        {
        if (*s1=='\0')
                return(0);
        }

return(-1);
}



produit(a,b,c,d)
char *a,*b,*c,**d;

/* calculer le produit de deux fnd codees dans les tableaux a et b
*/

{
int j,s;
char T[MAXSIZE],R[MAXSIZE],*p;

while((*a!='\0')&&(c!=*d))
        {
        p=b;
        for(j=0;j!=MAXSIZE;j++)
                R[j] = *a++;
        while((*p!='\0')&&(c!=*d))
                {
                for(j=0;j!=MAXSIZE;j++)
                        T[j] = R[j] & (*p++);
                s = simplification(&T[0]);
                if(s==0)
                        {
                        for(j=0;((j!=MAXSIZE)&&(c!=*d));j++)
                                *c++ = T[j];
                        }
                }
        }

if(c==*d)
        {
        printf(msgerreur[8]);
        erreur=1;
        }

*c = '\0';
*d = c;
}



PORMEE( )

/* analyser et coder une premisse existentielle;
*/

{
int j,k,l;
int val;
char T[MAXSIZE];

CONJONCTION( );
```

```
if (c!='(')
        {
        printf(msgerreur[3]);
        printf("%d\n", numpremisse);
        erreur = 1;
        }

if (erreur==0)
        {
        for(j=0;j!=MAXSIZE;j++)
                T[j] = ~0;
        while(pile!=NULL)
                {
                k = pile->nb;
                if (k < 0)
                     k = (-k);
                l = k - 1;
                k = k % 4;
                k = 8-(2*k);
                k = k % 8;
                l = l/4;
                if (pile->nb <0)
                     val = 1 << k;
                  else
                     val = 2 << k;
                val = ~val;
                T[l] = T[l]&val;
                pile = pile->next;
                }
        }

ELEMENTS(T);

if (c!=')')
        {
        printf(msgerreur[5]);
        printf("%d\n", numpremisse);
        erreur=1;
        }

c=lirecar(c);

}



ELEMENTS(d)
char *d;

/* analyser et memoriser une suite d'elements;
*/

{
int n,i,j;
char *q;
char nomdelt[11];

j=0;
c=lirecar(c);

while(j==0)
        {
        q = d;
        LECTURE(nomdelt);
        n=elttransf(nomdelt);
        if (n==(-1))
```

```
                {
                printf(msgerreur[7]);
                printf("%d\n", numpremisse);
                erreur=1;
                }

        if (erreur==0)
                {
                for(i=0;i!=MAXSIZE;i++)
                        {
                        codePE[n][i] = codePE[n][i] & (*q++);
                        }
                codePE[n][i] = '\0';
                }
        if (c==',')
                {
                c=lirecar(c);
                if (c != ' ')
                        {
                        erreur = 0;
                        printf(msgerreur[10]);
                        printf("%d\n",numpremisse);
                        }
                   else
                        c = lirecar(c);
                }
            else
                j=(-1);
        }
}


elttransf(a)
char *a;

/* attribuer un numero a un element;
*/

{
int i;

for (i=0;((i!=maxelt)&&(comparaison(&transfelt[i][0],a)!=0));i++);
if (i==MAXSIZE*24)
        i=(-1);
  else
        {
        if (i==maxelt)
                {
                copie(&transfelt[i][0],a);
                maxelt++;
                }
        }

return(i);
}


suppressionmltple(c,d,e,f)
char *c,*d,*e,**f;

/* supprimer les multiples d'une fnd = F+G, codee dans les
   tableaux c et d;
   G, codee dans d est deja debarassee de ses multiples;
*/
```

```
{
char *m,*p1,*p2,*q,*r,*z;
char L[MAXSIZE + 1],N[MAXSIZE + 1],PDT[MAXSIZE + 1];
int i;


m=c;

while(*m!='\0')
        {
        if (*m==' ')
                {
                for(i=0;i!=MAXSIZE;i++)
                        m++;
                }
            else
                {
                p1=m;
                for(i=0;i!=MAXSIZE;i++)
                        N[i] = *m++;
                N[i] = '\0';
                q=m;
                while (*q!='\0')
                        {
                        p2=q;
                        for (i=0;i!=MAXSIZE;i++)
                                {
                                L[i] = *q++;
                                PDT[i] = N[i] & L[i];
                                }
                        L[i] = '\0';
                        PDT[i] = '\0';
                        if ((comparaison(PDT,L))==0)
                                {
                                for (i=0;i!=MAXSIZE;i++)
                                        *p2++=' ';
                                }
                            else
                                {
                                if((comparaison(PDT,N))==0)
                                        {
                                        for (i=0;i!=MAXSIZE;i++)
                                                *p1++=' ';
                                        break;
                                        }
                                }
                        }
                 }
        }

m=d;

while (*m!='\0')
        {
        if (*m==' ')
                {
                for(i=0;i!=MAXSIZE;i++)
                        m++;
                }
             else
                {
                p1=m;
                for(i=0;i!=MAXSIZE;i++)
                        N[i] = *m++;
                N[i] = '\0';
                r=c;
```

```
                while(*r!='\0')
                        {
                        p2=r;
                        for (i=0;i!=MAXSIZE;i++)
                                {
                                L[i] = *r++;
                                PDT[i] = N[i] & L[i];
                                }
                        PDT[i]='\0';
                        L[i]='\0';
                        if (comparaison(PDT,L)==0)
                                {
                                for (i=0;i!=MAXSIZE;i++)
                                        *p2++=' ';
                                }
                            else
                                {
                                if ((comparaison(PDT,N))==0)
                                        {
                                        for(i=0;i!=MAXSIZE;i++)
                                                *p1++ = ' ';
                                        break;
                                        }
                                }
                        }
                }
        }

while((*c!='\0')&&(e!=*f))
        {
        if (*c!=' ')
                *e++ = *c;
        c++;
        }
z = e;

while((*d!='\0')&&(e!=*f))
        {
        if (*d!=' ')
                *e++ = *d;
        d++;
        }

if (e==*f)
        {
        printf(msgerreur[8]);
        erreur = 1;
        }

*e = '\0';
*f = z;
}



affichtermes(a)
char *a;

/* afficher a l'ecran une fnd codee dans le tableau a;
*/

{
char b,c;
int i,j,k,l;

l =0;
```

```
while(*a!='\0')
        {
        if (l != 0)
                {
                printf(" ");
                printf("+ \n");
                }
        l = 1;
        for(i=0;i!=MAXSIZE;i++)
                {
                c = *a++;
                j=0;
                while(j != 8)
                        {
                        b = c >> j;
                        b = b & 3;
                        k = 3 -(j/2) + (4*i);
                        if (b==1)
                             {
                             if (l!=1)
                                printf(" , ");
                             printf(transfterme[k]);
                             l=2;
                             }
                        if (b==2)
                             {
                             if (l!=1)
                                printf(" , ");
                             printf("-");
                             printf(transfterme[k]);
                             l = 2;
                             }
                        j+=2;
                        }
                }
         }
}


char *rechimplicants(a,y,acons,ncons,n)
char *a,*y,*acons,*ncons;
int n;

/* rechercher les implicants primitifs d'une fnd codee dans
   le tableau a
*/

{
char *p,*q,*r,cons[2*placereservee + 1];

r = cons;
p = r + (2*placereservee) + 1;
while((y!=a)&&(erreur==0))
         {
         consensus(a,y,cons,p);
         q = acons + n;
         suppressionmltple(cons,a,acons,&q);
         a = acons;
         y = q;
         acons = ncons;
         ncons = a;
         }

return(a);
}
```

```
consensus(a,y,cons,p)
char *a,*y,*cons,*p;

/* rechercher les consensus entre les monomes d'une fnd F+G;
   F est codee dans le tableau a;
   G est codee dans le tableau y;
   G ne contient deja que des implicants primitifs;
*/

{
char *m,*q,*c;
char N[MAXSIZE],L[MAXSIZE],R[MAXSIZE],PDT[MAXSIZE+1],t[MAXSIZE+1];
char masque[4];
int i,j,zero,n,k;


for(i=0;i!=4;i++)
        {
        j = i*2;
        j = 6 -j;
        masque[i] = 3 << j;
        }

for(i=0;i!=MAXSIZE;i++)
        t[i] = ~0;
t[i] = '\0';

c = cons;
m = a;

while((m != y)&&(erreur==0))
         {
         for(i=0;i!=MAXSIZE;i++)
                 N[i] = *m++;
         q = m;
         while((q != y)&&(erreur==0))
                 {
                 for(i=0;i!=MAXSIZE;i++)
                         {
                         L[i] = *q++;
                         PDT[i] = N[i] & L[i];
                         }
                 zero = 0;
                 for(i=0;((i!=MAXSIZE)&&(zero!=2));i++)
                         {
                         for(j=0;((j!=4)&&(zero!=2));j++)
                                 {
                                 R[i] = PDT[i] & masque[j];
                                 if (R[i]=='\0')
                                         {
                                         zero++;
                                         n=i;
                                         k = j;
                                         }
                                 }
                         }
                 if (zero==1)
                         {
                         PDT[n] = PDT[n] | masque[k];
                         PDT[MAXSIZE] = '\0';
                         if (comparaison(t,PDT)==0)
                                 {
                                 printf(msgerreur[13]);
                                 cons = c;
```

```
                        }
                    for(i=0;((i!=MAXSIZE)&&(cons!=p));i++)
                        *cons++ = PDT[i];

                    if (cons==p)
                        {
                        erreur = 1;
                        printf(msgerreur[8]);
                        }
                    }
                }
        }

m = y;
while((*m!='\0')&&(erreur==0))
        {
        for(i=0;i!=MAXSIZE;i++)
                N[i] = *m++;
        q = a;
        while((q != y)&&(erreur==0))
                {
                for(i=0;i!=MAXSIZE;i++)
                        {
                        L[i] = *q++;
                        PDT[i] = N[i] & L[i];
                        }
                zero = 0;
                for(i=0;((i!=MAXSIZE)&&(zero!=2));i++)
                        {
                        for(j=0;((j!=4) && (zero!=2));j++)
                                {
                                R[i] = PDT[i] & masque[j];
                                if (R[i] == '\0')
                                        {
                                        zero++;
                                        n = i;
                                        k = j;
                                        }
                                }
                        }
                if (zero == 1)
                        {
                        PDT[n] = PDT[n] | masque[k];
                        PDT[MAXSIZE] = '\0';
                        if(comparaison(t,PDT)==0)
                                {
                                printf(msgerreur[13]);
                                cons = c;
                                }
                        for(i=0;((i!=MAXSIZE)&&(cons!=p));i++)
                                *cons++ = PDT[i];
                        if (cons==p)
                                {
                                erreur = 1;
                                printf(msgerreur[8]);
                                }
                        }
                }
        }

*cons ='\0';
}
```

interro.c

```
# include <stdio.h>
# define MAXFORMAT 10
# define placereservee MAXFORMAT*100

FILE *fichier,*fopen( );

int erreur,c;

char codePE[MAXFORMAT*24][MAXFORMAT+1],
     codeinitial[2*placereservee + 1],
     contextecrt[2*placereservee + 1];

char transfterme[4*MAXFORMAT][10],
     transfelt[24*MAXFORMAT][10];

char zone1[2*placereservee + 1],
     zone2[2*placereservee + 1];

char *codecrt,*codepct,
     *ctxtcrt,*ctxtpct;

char quesconcl[placereservee + 1],
     queselt[placereservee + 1];


struct nbre
          {int nb;
           struct nbre *next;} *pile;

struct tableau
          {struct nbre *pt;
           struct tableau *suivant} *table;


char *msgerreur[13] =
     {"ce texte n'est pas code\n",
      "MAXSIZE est different de MAXFORMAT: modifier ce dernier\n",
      "il manque un signe d'implication '->'\n",
      "il manque un blanc apres le signe '->'\n",
      "il manque un '?'\n",
      "il manque un blanc apres le '?'\n",
      "il manque un blanc apres '>'\n",
      "cet element n'existe pas\n",
      "certains termes n'existent pas\n",
      "il manque un blanc apres '+'\n",
      "il manque un blanc apres ','\n",
      "pas assez de place: modifier placereservee\n",
      "il manque un '.'\n"
     };


main( )
{
char titre[12],x;
int i,j;

erreur = 1;
while(erreur==1)
          {
          i=0;
          printf("introduire le nom du texte :\n");
          while(((x=getchar( ))!='\n')&&(i!=9))
                    titre[i++] = x;
          while(x!='\n')
                    x=getchar( );
          titre[i++] = '.';
```

```
        titre[i++] = 'o';
        titre[i] = '\0';
        if ((fichier = fopen(titre,"r"))==NULL)
                printf(msgerreur[0]);
            else
                erreur = 0;
        }

x = getc(fichier);
if (x!=MAXFORMAT)
        afficherreur(1);

if (erreur == 0)
        {
        for(i=0;i!=MAXFORMAT*24;i++)
                for(j=0;j!=MAXFORMAT+1;j++)
                        codePE[i][j] = getc(fichier);

        i=0;
        while((x=getc(fichier))!='\0')
                codeinitial[i++] = x;
        codeinitial[i] = '\0';

        for(i=0;i!=4*MAXFORMAT;i++)
                {
                for(j=0;j!=10;j++)
                        transfterme[i][j] = getc(fichier);
                }

        for(i=0;i!=24*MAXFORMAT;i++)
                {
                for(j=0;j!=10;j++)
                        transfelt[i][j] = getc(fichier);
                }
        fclose(fichier);

        copie(zone1,codeinitial);
        codecrt = zone1;
        codepct = zone2;

        contextecrt[0] = '\0';
        ctxtcrt = &contextecrt[1];
        ctxtpct = ctxtcrt;

        putchar(27);
        putchar(61);
        putchar(32);
        putchar(32);
        putchar(27);
        putchar(89);

        erreur = 0;
        c = getchar();

        while (c!='$')
                {
                analyse();
                erreur = 0;
                c = getchar();
                }
        }
}


analyse()
{
```

```
char *x,*w,*z,res[MAXFORMAT + 1],*q;
int i,j,l;
char t[MAXFORMAT + 1],resultat[2*placereservee + 1],
     ac[2*placereservee + 1],nc[2*placereservee + 1];
char *r,*s,*p,y;

switch(c) {
        case '\n': break;

        case 9   : break;

        case '*': {
                  affichtermes(codecrt);
                  break;
                  }

        case ':': {
                  copie(zone1,codeinitial);
                  codecrt = zone1;
                  codepct = zone2;
                  contextecrt[0] = '\0';
                  ctxtcrt = &contextecrt[1];
                  ctxtpct = ctxtcrt;
                  break;
                  }

        case '<': {
                  x = codecrt;
                  codecrt = codepct;
                  codepct = x;
                  ctxtcrt = ctxtpct;
                  break;
                  }

        case '>': {
                  demelt( );
                  if (erreur ==0)
                        {
                        for (i=0;i!=24*MAXFORMAT;i++)
                                {
                                q = queselt;
                                for (j=0;j!=MAXFORMAT;j++)
                                        res[j] = codePE[i][j] & *q++;
                                res[j] = '\0';
                                if(comparaison(codePE[i],res)==0)
                                        {
                                        printf(transfelt[i]);
                                        printf("   ");
                                        }
                                }
                        }
                  break;
                  }

        case '?': {
                  for (i=0;i!=MAXFORMAT;i++)
                          t[i] = ~0;
                  t[i] = '\0';
                  i = demterme( );
                  if (i == -1)
                          afficherreur(7);
                  if (erreur == 0)
                          {
                          if (comparaison(&codePE[i][0],t) == 0)
                              afficherreur(7);
                            else
```

```
                                affichtermes(codePE[i]);
                            }
                    break;
                    }

        case '#' : {
                    affich(contextecrt);
                    break;
                    }

        default : {
                    w = ctxtcrt;
                    z = ctxtpct;
                    demconclusion( );
                    if (erreur == 1)
                          {
                          ctxtcrt = w;
                          ctxtpct = z;
                          }

                    if (erreur == 0)
                          {
                          for (i=0;i!=MAXFORMAT;i++)
                                  t[i] = '0';
                          t[i] = '\0';
                          r = resultat + 2*placereservee +1;
                          produit(t,quesconcl,resultat,&r);

                          l = 2*placereservee + 1;
                          p = rechimplicants(resultat,r,ac,nc,l);

                          r = resultat + 2*placereservee + 1;
                          produit(p,codecrt,resultat,&r);

                          y = '\0';
                          s = &y;
                          r = codecrt + 2*placereservee + 1;
                          suppressionmltple(resultat,s,codepct,&r);
                          x = codepct;
                          codepct = codecrt;
                          codecrt = x;

                          examen(codecrt);
                          }
                    break;
                    }
        }
while(c != '\n')
        c = getchar( );
}

demtermes( )
{
char nom[10];
int i;

c = getchar( );
i = -1;

if (c != ' ')
        afficherreur(5);
    else
        {
        c = getchar( );
        lecture(nom);
        if (c != '.')
```

```
                afficherreur(12);
            else
                i = elttransf(nom);
        }

return(i);
}


demelt( )
{
char *r;
int k;
struct fonction *t,*calloc( );
c = getchar( );
if (c != ' ')
        afficherreur(6);

if (erreur == 0)
        {
        c = getchar( );
        k = 1;
        conjonction(k);
        if (c != '?')
                afficherreur(4);
            else
                {
                t = calloc(1,sizeof(struct tableau));
                t->pt = pile;
                t->suivant = table;
                table = t;
                r = queselt + placereservee + 1;
                coderp(queselt,&r);
                }
        }
}

demconclusion( )
{
char *r;

fnd( );

if (erreur == 0)
        {
        if (c!= '-')
                afficherreur(2);
            else
                {
                c = getchar( );
                if (c!= '>')
                    afficherreur(2);
                }
        }
if (erreur == 0)
        {
        r = quesconcl + placereservee + 1;
        coderp(quesconcl,&r);
        }
 }


examen(a)
char *a;

{
```

```
char termecommun[MAXFORMAT + 1],
     t[MAXFORMAT + 1];

int i,j;

for(i=0;i!=MAXFORMAT;i++)
        t[i]=~0;
t[i] = '\0';

if(*a=='\0')
        printf("IMPOSSIBLE DANS CE CONTEXTE\n");
    else
        {
        if (comparaison(a,t)==0)
              printf("TAUTOLOGIE\n");
           else
              {
              for(i=0;i!=MAXFORMAT;i++)
                     termecommun[i] = *a++;
              while(*a!='\0')
                     {
                     for(i=0;i!=MAXFORMAT;i++)
                            termecommun[i] = termecommun[i] | *a++;
                     }
              termecommun[i] = '\0';

              if (comparaison(termecommun,t)==0)
                     printf("- \n");
                 else
                     affichtermes(termecommun);

              }
        }
}

afficherreur(i)
int i;
{
printf(msgerreur[i]);
erreur = 1;

}

affichtermes(a)
char *a;

{
char b,c;
int i,j,k,l;

l=0;
while(*a!='\0')
        {
        if(l != 0)
               {
               printf(" ");
               printf(" +  ");
               }
        l = 1;
        for(i=0;i!=MAXFORMAT;i++)
               {
               c = *a++;
               j = 0;
               while(j != 8)
                      {
                      b = c >> j;
```

```
                        b = b & 3;
                        k = 3 - (j/2) + (4*i);
                        if (b==1)
                                {
                                if (l!=1)
                                    printf(" , ");
                                printf(transfterme[k]);
                                l = 2;
                                }
                        if (b==2)
                                {
                                if (l!=1)
                                    printf(" , ");
                                printf("-");
                                printf(transfterme[k]);
                                l = 2;
                                }
                        j+=2;
                        }
                }
        }
}

copie(s1,s2)
char *s1,*s2;
{
while(*s1++ = *s2++);
}



fnd( )


{
struct tableau *t,*calloc( );
int i,k;
char *x;

x = ctxtcrt;

table = NULL;

k = 0;
i = 0;
while(i==0)
        {
        conjonction(k);

        if (erreur==0)
                {
                *ctxtcrt++ = '\0';
                t=calloc(1,sizeof(struct tableau));
                t->pt=pile;
                t->suivant=table;
                table=t;
                }

        if(c=='+')
                {
                c = getchar( );
                if ( c!= ' ')
                        afficherreur(9);
                    else
                        c = getchar( );
                }
```

```
            else
                i=(-1);
        }
if (erreur == 0)
        {
        *ctxtcrt++ = '\0';
        ctxtpct = x;
        }

}



conjonction(k)
int k;

{
struct nbre *p,*calloc( );
int i,nombre;

i=0;
pile=NULL;
while (i==0)
        {
        nombre=mot( );

        if (k == 0)
                {
                if (ctxtcrt == contextecrt + 2*placereservee)
                        afficherreur(11);
                }

        if (erreur==0)
                {
                if (k == 0)
                        *ctxtcrt++ = nombre;
                p=calloc(1,sizeof(struct nbre));
                p->nb=nombre;
                p->next=pile;
                pile=p;
                }

        if (c==',')
                {
                c = getchar( );
                if (c != ' ')
                        afficherreur(10);
                    else
                        c = getchar( );
                }
           else
                i=(-1);
        }
}



coderp(f,g)
char *f,**g;


{
int val;
char T[MAXFORMAT];
int j,k,l;
```

```
while ((table!=NULL)&&(f!=*g))
        {
        for(j=0;j!=MAXFORMAT;j++)
                T[j] = ~0;

        while(table->pt != NULL)
                {
                k=table->pt->nb;
                if (k < 0)
                     k =(-k);
                l = k - 1;
                l = l/4;
                k = k % 4;
                k = 8-(2*k);
                k = k % 8;
                if (table->pt->nb < 0)
                        val = 1 << k;
                   else
                        val = 2 << k;
                val = ~val;
                T[l]=T[l] & val;
                table->pt = table->pt->next;
                }

        for(j=0;((j!=MAXFORMAT)&&(f!=g[0]));j++)
                *f++ = T[j];

        table = table->suivant;
        }

if (f==*g)
        afficherreur(11);

*f = '\0';
*g = f;

}


int mot( )


{
int i;
char nomdeterme[10];

if (c=='-')
         {
         c = getchar( );
         nomdeterme[0] = '\0';
         lecture(nomdeterme);

         i=transf(nomdeterme);
         i = (-i);
         if (i==1)
               afficherreur(8);
         }
    else
         {
         lecture(nomdeterme);
         i=transf(nomdeterme);
         if (i==-1)
               afficherreur(8);
         }

return(i);
```

```
}


lecture(d)
char *d;


{
int e,f,g,h,i;

i=0;
e = (c==' ');
f = ((c>='0')&&(c<='9'));
g = ((c>='A')&&(c<='Z'));
h = ((c>='a')&&(c<='z'));
while ((e || f || g || h)&&(i!=9))
        {
        d[i++] = c;
        c = getchar();
        e = (c==' ');
        f = ((c>='0')&&(c<='9'));
        g = ((c>='A')&&(c<='Z'));
        h = ((c>='a')&&(c<='z'));
        }
d[i] = '\0';

if (i==9)
        {
        while (e || f || g || h)
                {
                c = getchar();
                e = (c==' ');
                f = ((c>='0')&&(c<='9'));
                g = ((c>='A')&&(c<='Z'));
                h = ((c>='a')&&(c<='z'));
                }
        }

}


transf(d)
char *d;


{
int i;

for(i=0;((i!=MAXFORMAT*4)&&(comparaison(transfterme[i],d)!=0));i++);
if (i== MAXFORMAT*4)
       i=(-2);
i++;
return(i);
}


simplification(k)
char *k;


{
int i,j,s,val;
```

```
s=0;
for(j=0;j!=MAXFORMAT;j++)
        {
        i=0;
        val=1;
        while ((i!=8)&&(val!=0))
                {
                val = (*(k+j) >> i);
                val = val & 3;
                i=i+2;
                }
        if (val==0)
                {
                s=(-1);
                break;
                }
        }

return(s);
}


comparaison(s1,s2)
char *s1,*s2;


{
for(;*s1==*s2;s1++,s2++)
        {
        if (*s1=='\0')
                return(0);
        }

return(-1);
}



produit(a,b,c,d)
char *a,*b,*c,**d;

{
int i,j,s;
char T[MAXFORMAT],R[MAXFORMAT],*p;

while((*a!='\0')&&(c!=*d))
        {
        p=b;
        for(j=0;j!=MAXFORMAT;j++)
                R[j] = *a++;
        while((*p!='\0')&&(c!=*d))
                {
                for(j=0;j!=MAXFORMAT;j++)
                        T[j] = R[j] & (*p++);
                s = simplification(&T[0]);
                if(s==0)
                        {
                        for(j=0;((j!=MAXFORMAT)&&(c!=*d));j++)
                                *c++ = T[j];
                        }
                }
        }

if(c==*d)
        afficherreur(11);
```

```
*c = '\0';
*d = c;
}


elttransf(a)
char *a;


{
int i;

for (i=0;((i!=MAXFORMAT*24)&&(comparaison(transfelt[i],a)!=0));i++);
if (i==MAXFORMAT*24)
        i=(-1);

return(i);
}


suppressionmltple(c,d,e,f)
char *c,*d,*e,**f;


{
char *m,*p1,*p2,*q,*r,*z;
char L[MAXFORMAT + 1],N[MAXFORMAT + 1],PDT[MAXFORMAT + 1];
int i;


m=c;

while(*m!='\0')
        {

        if (*m==' ')
                {
                for(i=0;i!=MAXFORMAT;i++)
                        m++;
                }
            else
                {
                p1=m;
                for(i=0;i!=MAXFORMAT;i++)
                        N[i] = *m++;
                N[i] = '\0';
                q=m;
                while (*q!='\0')
                        {
                        p2=q;
                        for (i=0;i!=MAXFORMAT;i++)
                                {
                                L[i] = *q++;
                                PDT[i] = N[i] & L[i];
                                }
                        L[i] = '\0';
                        PDT[i] = '\0';
                        if ((comparaison(PDT,L))==0)
                                {
                                for (i=0;i!=MAXFORMAT;i++)
                                        *p2++=' ';
                                }
                            else
                                {
                                if((comparaison(PDT,N))==0)
                                        {
```

```
                                        for (i=0;i!=MAXFORMAT;i++)
                                                *p1++=' ';
                                        break;
                                        }
                                }
                        }
                }
        }

m=d;

while (*m!='\0')
        {
        if (*m==' ')
                {
                for(i=0;i!=MAXFORMAT;i++)
                        m++;
                }
            else
                {
                p1=m;
                for(i=0;i!=MAXFORMAT;i++)
                        N[i] = *m++;
                N[i] = '\0';
                r=c;
                while(*r!='\0')
                        {
                        p2=r;
                        for (i=0;i!=MAXFORMAT;i++)
                                {
                                L[i] = *r++;
                                PDT[i] = N[i] & L[i];
                                }
                        PDT[i]='\0';
                        L[i]='\0';
                        if (comparaison(PDT,L)==0)
                                {
                                for (i=0;i!=MAXFORMAT;i++)
                                        *p2++=' ';
                                }
                            else
                                {
                                if ((comparaison(PDT,N))==0)
                                        {
                                        for(i=0;i!=MAXFORMAT;i++)
                                                *p1++ = ' ';
                                        break;
                                        }
                                }
                        }
                }
        }

while((*c!='\0')&&(e!=*f))
        {
        if (*c!=' ')
            *e++ = *c;
        c++;
        }
z = e;

while((*d!='\0')&&(e!=*f))
         {
         if (*d!=' ')
              *e++ = *d;
         d++;
```

```
        }

if (e==*f)
        afficherreur(11);

*e = '\0';
*f = z;
}



char *rechimplicants(a,y,acons,ncons,n)
char *a,*y,*acons,*ncons;
int n;


{
char *p,*q,*r,cons[2*placereservee + 1];

r = cons;
p = r + (2*placereservee) + 1;
while((y!=a)&&(erreur==0))
        {
        consensus(a,y,cons,p);
        q = acons + n;
        suppressionmltple(cons,a,acons,&q);
        a = acons;
        y = q;
        acons = ncons;
        ncons = a;
        }

return(a);
}



consensus(a,y,cons,p)
char *a,*y,*cons,*p;


{
char *m,*q,*c;
char N[MAXPORMAT],L[MAXPORMAT],R[MAXPORMAT],
     PDT[MAXPORMAT + 1],t[MAXPORMAT + 1];
char masque[4];
int i,j,zero,n,k;


for(i=0;i!=4;i++)
        {
        j = i*2;
        j = 6 -j;
        masque[i] = 3 << j;
        }

for(i=0;i!=MAXPORMAT;i++)
        t[i] = ~0;
t[i] = '\0';

c = cons;
m = a;

while((m != y)&&(erreur==0))
        {
        for(i=0;i!=MAXPORMAT;i++)
```

```
                N[i] = *m++;
        q = m;
        while((q != y)&&(erreur==0))
                {
                for(i=0;i!=MAXFORMAT;i++)
                        {
                        L[i] = *q++;
                        PDT[i] = N[i] & L[i];
                        }
                zero = 0;
                for(i=0;((i!=MAXFORMAT)&&(zero!=2));i++)
                        {
                        for(j=0;((j!=4)&&(zero!=2));j++)
                                {
                                R[i] = PDT[i] & masque[j];
                                if (R[i]=='\0')
                                        {
                                        zero++;
                                        n=i;
                                        k = j;
                                        }
                                }
                        }
                if (zero==1)
                        {
                        PDT[n] = PDT[n] | masque[k];
                        PDT[MAXFORMAT] = '\0';
                        if (comparaison(t,PDT)==0)
                              cons = c;

                        for(i=0;((i!=MAXFORMAT)&&(cons!=p));i++)
                              *cons++ = PDT[i];

                        if (cons==p)
                              afficherreur(11);
                        }
                }
        }

m = y;
while((*m!='\0')&&(erreur==0))
        {
        for(i=0;i!=MAXFORMAT;i++)
                N[i] = *m++;
        q = a;
        while((q != y)&&(erreur==0))
                {
                for(i=0;i!=MAXFORMAT;i++)
                        {
                        L[i] = *q++;
                        PDT[i] = N[i] & L[i];
                        }
                zero = 0;
                for(i=0;((i!=MAXFORMAT)&&(zero!=2));i++)
                        {
                        for(j=0;((j!=4) && (zero!=2));j++)
                                {
                                R[i] = PDT[i] & masque[j];
                                if (R[i] == '\0')
                                      {
                                      zero++;
                                      n = i;
                                      k = j;
                                      }
                                }
                        }
```

```
                if (zero == 1)
                        {
                        PDT[n] = PDT[n] | masque[k];
                        PDT[MAXFORMAT] = '\0';
                        if(comparaison(t,PDT)==0)
                                cons = c;

                        for(i=0;((i!=MAXFORMAT)&&(cons!=p));i++)
                                *cons++ = PDT[i];
                        if (cons==p)
                                afficherreur(11);
                        }
                }
        }

*cons ='\0';
}

affich(a)
char *a;
{
int i;

if (ctxtcrt == &contextecrt[1])
        printf("contexte initial\n");
while(a != ctxtcrt - 1)
        {
        if (*a == '\0')
                {
                printf("->");
                a++;
                }
        while( *a != '\0')
                {
                i = *a++;
                if (i < 0)
                        {
                        printf("-");
                        i = -i;
                        }
                i = i -1;
                printf(transfterme[i]);
                printf("  ");
                }
        a++;
        if (*a != '\0')
                printf("+");
        }
}
```

TABLE DES MATIERES